KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Mathematical Statistics

Second cycle, 30 credits

Balancing Performance and Usage Cost: A Comparative Study of Language Models for Scientific Text Classification

EVA ENGEL

Stockholm, Sweden, 2023

Balancing Performance and Usage Cost: A Comparative Study of Language Models for Scientific Text Classification

EVA ENGEL

Master's Programme, Applied and Computational Mathematics,
120 credits
Date: September 1, 2023
Supervisor: Joakim Nivre, Johan Broberg, Jimmy Olsson
Examiner: Jimmy Olsson
School of Electrical Engineering and Computer Science
Host company: RISE Research Institutes of Sweden AB
Swedish title: Balansera prestanda och användningskostnader: En jämförande undersökning av språkmodeller för klassificering av vetenskapliga texter

Balancing Performance and Usage Cost: A Comparative Study of Language Models for Scientific Text Classification / Balansera prestanda och användningskostnader: En jämförande undersökning av språkmodeller för klassificering av vetenskapliga texter

Abstract

The emergence of large language models, such as BERT and GPT-3, has revolutionized natural language processing tasks. However, the development and deployment of these models pose challenges, including concerns about computational resources and environmental impact. This study aims to compare discriminative language models for text classification based on their performance and usage cost. We evaluate the models using a hierarchical multi-label text classification task and assess their performance using primarly F1-score. Additionally, we analyze the usage cost by calculating the Floating Point Operations (FLOPs) required for inference. We compare a baseline model, which consists of a classifier chain with logistic regression models, with fine-tuned discriminative language models, including BERT with two different sequence lengths and DistilBERT, a distilled version of BERT. Results show that the DistilBERT model performs optimally in terms of performance, achieving an F1-score of 0.56 averaged on all classification layers. The baseline model and BERT with a maximal sequence length of 128 achieve F1-scores of 0.51. However, the baseline model outperforms the transformers at the most specific classification level with an F1-score of 0.33. Regarding usage cost, the baseline model significantly requires fewer FLOPs compared to the transformers. Furthermore, restricting BERT to a maximum sequence length of 128 tokens instead of 512 sacrifices some performance but offers substantial gains in usage cost. The code and dataset are available on GitHub.[1]

Keywords

Natural language processing; hierarchical multi-label text classification; BERT; comparative study; floating point operations

[1] https://github.com/eengel7/comparison_NLP_classification_models

Sammanfattning

Balansera prestanda och användningskostnader: En jämförande undersökning av språkmodeller för klassificering av vetenskapliga texter

Uppkomsten av stora språkmodeller, som BERT och GPT-3, har revolutionerat språkteknologi. Dock ger utvecklingen och implementeringen av dessa modeller upphov till utmaningar, bland annat gällande beräkningsresurser och miljöpåverkan. Denna studie syftar till att jämföra diskriminativa språkmodeller för textklassificering baserat på deras prestanda och användningskostnad. Vi utvärderar modellerna genom att använda en hierarkisk textklassificeringsuppgift och bedöma deras prestanda primärt genom F1-score. Dessutom analyserar vi användningskostnaden genom att beräkna antalet flyttalsoperationer (FLOPs) som krävs för inferens. Vi jämför en grundläggande modell, som består av en klassifikationskedja med logistisk regression, med finjusterande diskriminativa språkmodeller, inklusive BERT med två olika sekvenslängder och DistilBERT, en destillerad version av BERT. Resultaten visar att DistilBERT-modellen presterar optimalt i fråga om prestanda och uppnår en genomsnittlig F1-score på 0,56 för alla klassificeringsnivåer. Den grundläggande modellen och BERT med en maximal sekvenslängd på 128 uppnår ett F1-score på 0,51. Dock överträffar den grundläggande modellen transformermodellerna på den mest specifika klassificeringsnivån med en F1-score på 0,33. När det gäller användningskostnaden kräver den grundläggande modellen betydligt färre FLOPs jämfört med transformermodellerna. Att begränsa BERT till en maximal sekvenslängd av 128 tokens ger vissa prestandaförluster men erbjuder betydande besparingar i användningskostnaden. Koden och datamängden är tillgängliga på GitHub.[1]

Nyckelord

Språkteknologi; hierarkisk textklassificering; BERT; jämförande studie; flyttalsoperationer

[1] https://github.com/eengel7/comparison_NLP_classification_models

Acknowledgments

I would like to express my heartfelt gratitude to everyone who has contributed to my research and to this work. First and foremost, I would like to thank RISE, Research Institutes of Sweden, for the opportunity to work on this research project and the enriching time I spent there. A special thank you goes to my supervisor Joakim Nivre for his versatile expertise, insights and feedback. I am also grateful to my additional supervisor and mentor Johan Broberg for his continuous support, his diverse feedback, and for being a “bollplank”. Additionally, I would like to acknowledge Joakim Eriksson for establishing the connection to Ymner. I thank Ymner and especially Oscar Ridell for providing the data, the original idea and the freedom to combine my interests. I would also like to thank my supervisor Jimmy Olsson at KTH Royal Institute of Technology for his valuable feedback and guidance throughout this research journey. Lastly, I am deeply grateful to all my friends and my family for their unwavering support and encouragement throughout this endeavor.

Stockholm, September 2023
Eva Engel

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

AI Artificial Intelligence

AUC Area under ROC Curve

BERT Bidirectional Encoder Representations from Transformers

BOW Bag-of-Words

CPU Central Processing Unit

DeBERTa Decoding-enhanced BERT with disentangled attention

DistilBERT DistilBERT

DL Deep Learning

FinBERT FinBERT

FLOPs Number of Floating-Point Operations

FN False Negatives

FNN Feedforward Neural Network

FP False Positives

GLUE General Language Understanding Evaluation

GPT Generative Pre-Trained Transformer

GPU Graphics Processing Unit

GRU Gated Rectified Unit

HBGL Hierarchy-guided BERT with Global and Local Hierarchies

HGCLR Hierarchy-guided Contrastive Learning

HMTC Hierarchical Multi-Label Text Classification

LLM Large Language Model

LRAP Label Ranking Average Precision

LSTM Long Short-Term Memory

ML Machine Learning

MLM Masked Language Modeling

NLP Natural Language Processing

NSP Next Sentence Prediction

PLM Pre-trained Language Model

RNN Recurrent Neural Network

RoBERTa Robustly Optimized BERT Approach

ROC Receiver Operating Characteristics

SCB Statistiska Centralbyrån

SVM Support Vector Machine

TF-IDF Term Frequency-Inverse Document Frequency

TN True Negatives

TP True Positives

TPU Tensor Processing Unit

ULMFiT Universal Language Model Fine-tuning

XLM XLM

XLNet XLNet

XMTC Extreme Multi-label Text Classification

Chapter 1

Introduction

1.1 Motivation

The field of Natural Language Processing (NLP) has undergone a paradigm shift in recent years with the emergence of Large Language Models (LLMs) such as Bidirectional Encoder Representations from Transformers (BERT), Generative Pre-Trained Transformer (GPT)-3, ChatGPT, and GPT-4. These models have achieved unprecedented levels of performance across a wide range of natural language tasks, including text classification, machine translation, question-answering, and text generation. The success of LLMs is primarily attributed to the convergence of three technical forces: advances in hardware technology, algorithmic approaches, and the availability of large-scale datasets. With the availability of large amounts of computing resources and data, researchers have been able to train LLMs with billions of parameters, leading to significant improvements in natural language processing.

The above mentioned LLMs are based on the transformer architecture, which is a type of neural network introduced in Vaswani et al. in 2017 [56]. It has since become the cornerstone of many state-of-the-art NLP models. The core idea behind transformers is the use of self-attention mechanisms, which allow the model to selectively focus on different parts of the input sequence and output of the decoders when generating output. This makes transformers particularly effective at modeling long-range dependencies in text. To train transformers, researchers typically use a two-step process known as pre-training and fine-tuning. During pre-training, the model is trained on massive amounts of unlabeled text data, such as books, articles, and web pages. This process is known as self-supervised learning, and the goal is to teach the model to understand the underlying structure and patterns of

natural language. Once the model has been pre-trained, it can be fine-tuned on specific NLP tasks, such as text classification, using a smaller labeled dataset. The fine-tuning process allows the model to adapt to the specific task and improve its performance. In recent years, the availability of large-scale datasets, such as Common Crawl,[1] allowed models to capture a vast amount of linguistic knowledge and generalize well to new tasks. Additionally, advances in hardware, such as Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs), have made it possible to train these massive models in a reasonable amount of time.

Despite their impressive performance, the development of LLMs also presents challenges and concerns, including computational resources and the environmental impact. The amount of computation used to train the largest deep learning models has increased 300,000 times in the years 2014 to 2020, increasing at a pace that far exceeds Moore's Law [50]. In addition to training, storage and model inference can impose significant costs, raising questions about their sustainability in the long term. This is especially true for models with billions of parameters, such as GPT-3, which can require up to hundreds of gigabytes of memory for inference. Patterson et al. [41] analyzed the energy consumption of training and running an NLP model and calculated the carbon footprint as the composition of the electrical energy from training but also the number of queries along with the energy for inference. In 2019, Nvidia CEO Jensen Huang stated that 80 to 90 percent of the cost of Machine Learning (ML) at scale is devoted to Artificial Intelligence (AI) inference [29]. Similarly, Amazon Web services claimed that inferencing can account for up to 90% of the cost of their Machine Learning (ML) work [5]. While this might not be fully accurate for the services of the discriminative LLMs like BERT, inference costs are nevertheless an essential component in the usage cost and thus in the study of LLM.

As awareness of sustainable model design steadily grows, there is a need to compare language models regarding not only their performance but also their usage cost. A comparative study can promote the development of models that strike a balance between performance and usage cost. Furthermore, it enables us to select the most appropriate model for a given NLP task, depending on its requirements and constraints.

In the work by Galke et al. [22], a detailed comparison of sequence-based classifiers in multi-label classification is provided. They investigate various models, including BERT (base and large), DistilBERT, Robustly Optimized BERT Approach (RoBERTa), and Decoding-enhanced BERT with

[1] https://commoncrawl.org

disentangled attention (DeBERTa), along with traditional methods like Bag-of-Words (BOW) and graph-based models. Their analysis primarily relies on the number of model parameters as an indicator of efficiency. However, it is important to create a more robust knowledge of the efficiency of the models by incorporating additional metrics. We propose using FLOPs, which is a measure of the number of floating point operations.

1.2 Research Questions and Objectives

This thesis aims to conduct a comprehensive comparison of discriminative language models for the NLP task of text classification. On the one hand, we aim to evaluate the performance of various models by measuring the quality of their predictions against ground truth data. To achieve this, we will assess the models on multiple metrics using a test data set. On the other hand, we analyze the usage cost for inference by calculating FLOPs, which is a measure of the number of floating point operations required for a given model to perform inference [22].

The data set provided by Ymner [1] represents an Hierarchical Multi-Label Text Classification (HMTC) task. The objective is to classify research projects based on their titles and descriptions and assign them to the corresponding research areas. These research areas are organized hierarchically in a tree-like structure that consists of multiple levels of specifications. For instance, a research project labeled as *Geometry* at the most specific classification level can also be inferred to belong to *Mathematics* and subsequently to *Natural Science*, given the hierarchical taxonomy. Therefore, the task at hand involves a multi-label classification. Additionally, research projects may be associated with multiple research areas, allowing them to follow multiple paths in the hierarchical classification tree. Returning to our example, a research project may address both *Geometry* and *Biophysics*, which can be found by following distinct paths in the classification structure.

For the comparative study, we will evaluate a baseline model based on traditional ML and discriminative language models. For the latter one, we will use pre-trained checkpoints of BERT for two maximal sequence lengths and DistilBERT (DistilBERT), a distilled version of BERT. In the whole process of deploying classification models and transformers, in particular, there are many computational parts for which one can calculate FLOPs. Since we use pre-trained language models, it is not feasible in the context of this work to

[1] https://www.ymner.com

compute FLOPs for training. Computing FLOPs for fine-tuning could also be an option, however, we focus on the inference cost. As stated above, it contributes to the overall cost of a model especially with regard to long-term deployments. Additionally, this way we can compare it to the baseline, which does not rely on fine-tuning.

In summary, we will implement classification models, define a fair and comparable evaluation framework, and benchmark performance against usage cost. Consequently, the following research questions arise:

RQ1 How can we classify scientific text given a hierarchical classification structure?

RQ2 How can we compare the usage cost of the models in our classification setting?

RQ3 Which classification model performs optimally considering the performance and usage cost?

The results reveal that while the baseline model is computationally efficient, the transformer models achieve better performance for the overall classification task involving all classification levels. Among the transformers, DistilBERT performs particularly well, although the baseline model outperforms the transformers in the most specific classification level. The FLOPs analysis shows that the baseline model is computationally efficient, while transformer models require significantly higher computational resources. The analysis indicates that transformer models excel in broader classifications but struggle with more detailed ones, suggesting that they may face challenges in leveraging subtle distinctions between labels.

1.3 Course of Investigation

In order to answer the research questions of this thesis, we will pursue the following course of investigation. In Chapter 2, we start by giving an overview of the theoretical concept and techniques of text classification along with the foundation of transformers. Here, we focus on the first research question **RQ1**. Then, we will describe metrics to evaluate the performance and usage cost, where we partly address research question **RQ2**. Chapter 3 presents the state-of-the-art technologies and approaches in hierarchical multi-label text classification and comparison of transformers. In Chapter 4, we will describe our proposed method and its application to the data set. Thereby,

we answer research question **RQ2**. Chapter 5 serves to evaluate and discuss our approach. We will conclude research question **RQ1** and focus on the optimization problem regarding research question **RQ3**. Finally, Chapter 6 summarises the work and shows the impact of this thesis on the field of research. Furthermore, we will give a short outlook on further ideas in this area of research.

Chapter 2

Background

This chapter is devoted to clarifying the fundamental concepts of classification models in NLP. First, we describe text classification with a focus on HMTC. Second, we look at ML for text classification and present a multi-label logistic regression model for text classification that will serve as a baseline model. Then Section 2.3 describes Deep Learning (DL) based approaches and models. Finally, Section 2.5 outlines metrics that can be used to evaluate the performance and usage cost of classification models.

2.1 Text Classification

Text classification or text categorization is a classical problem in NLP that aims to assign labels to textual units [35]. A textual unit can represent, for example, a document, a sentence, a query, or a paragraph, but for clarity, we will mainly refer to documents. In traditional single-label classification, a document is assigned a single label from a set of disjoint classes $\mathbb{C} = \{c_1, c_2, ...\}$. In other words, given a description $d \in \mathbb{X}$ of a document with $\mathbb{X}$ being the document space, a labeled document is described as $\langle d, c \rangle \in \mathbb{X} \times \mathbb{C}$. A so-called *soft version* of the classification problem assigns a probability value to each label instead of the label itself [4]. If $|\mathbb{C}| = 2$, we call it a binary classification problem, and if $|\mathbb{C}| > 2$ a multi-class classification problem. In contrast, multi-label classification describes the case where documents are associated with a set of labels $L \subseteq \mathbb{C}$ [54].

2.1.1 Hierarchical Multi-Label Text Classification

HMTC is a special case of text classification where the classes are organized in a taxonomic hierarchy. This taxonomic hierarchy is usually modeled as a tree or a directed acyclic graph, where each node represents a label to be classified [60]. As we move down the taxonomic hierarchy towards the leaves, the classes become more specific, while moving up towards the root, the classes become broader in scope. This means that classes are structured in subsets until a certain level at which there is no further specification. Consequently, we are dealing with a multi-label classification setting where the labels have dependencies. Figure 2.1 illustrates part of a hierarchical classification structure for labeling a document according to its research area. Suppose that a document belongs to the label *Geometry*, we can conclude that it also belongs to *Mathematics* and subsequently to *Natural Science*. This is called a single-path classification which directly implies a multi-label setup. In addition to that, one speaks of multi-path labels if a document has multiple labels within the same node level and thus follows several paths in the classification tree. For example, a document can belong both to *Geometry* and *Organic Chemistry* (and their parent nodes).

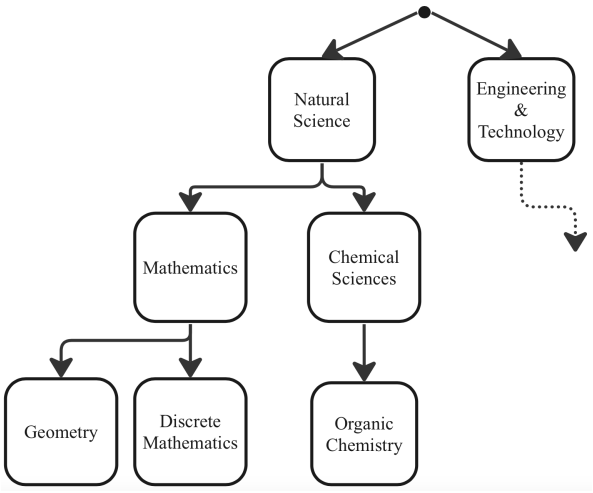


Figure 2.1: Example of a hierarchical classification structure that includes different levels of research fields.

The taxonomic hierarchy models dependencies among the labels. Apart from the trivial vertical correlations (e.g. *Natural Sciences* and *Mathematics*), there can also exist horizontal correlations that could reflect the relationships at the same level (e.g. *Geometry* and *Discrete Mathematics*). Any HMTC

problem can be easily modified to a flat multi-label problem, that ignores the vertical correlations. This implies that we are dealing with a set of labels that contains all the nodes of the graph [62].

2.2 Machine Learning in Text Classification

The general idea of ML is to learn meaningful representations of the input data by being exposed to known examples of inputs and outputs. In NLP, this input data consists of natural language. It combines computational linguistics—modeling of human language—with ML, statistical learning, and Deep Learning (DL). For automatic text classification, there are two categories of approaches: rule-based methods and ML-based (data-driven) methods [37]. Rule-based methods require deep domain knowledge to define a set of rules that determine the label of a document. In contrast, ML-based methods utilize a learning method or algorithm. Based on observations of data, a classifier or classification function γ learns to map documents to their corresponding classes:

$$\gamma : \mathbb{X} \rightarrow \mathbb{C}.$$

For labeled training data $\mathbb{D}$, this type of learning is called supervised learning. The supervised learning method is denoted by Γ and we define $\Gamma(\mathbb{D}) = \gamma$ where the learning method takes the training data set as input and returns the learned classification function [35].

2.2.1 Feature Extraction

Traditional ML-based models often follow a two-step procedure of feature extraction and prediction. First, hand-crafted features must be extracted from the documents, which are then passed to a classifier to predict the label or classification probability.

The idea of feature extraction is to convert the document, which is initially in a string format, into processable features that represent the document. Therefore, we first break down the document into units that are called tokens. Note that there are several types of tokenizers. Tokenizers often used with transformers tend to utilize different methods to assign a unique numerical token, allowing them to add special tokens and handle rare words or words not included in the vocabulary. However, traditional ML models typically use less complex approaches to tokenize a document such as dividing it into words, characters, or also n-grams. The latter representation divides a document

into word sequences of length n such that the resulting units overlap. Units can either be stored in a sequence and thus in an order-preserving list or in a set. To remove noise and redundant information from tokens, one can apply techniques to normalize and clean them. Besides converting text to lowercase or removing stop words (words that are commonly used due to the nature of language), one can also apply stemming or lemmatization. Stemming is a heuristic algorithm that truncates the suffixes or prefixes of a word to convert them into shorter stems and eliminate redundancy. For example, the words *fishing*, *fished*, and *fisher* would be converted to the base *fish*. In contrast, lemmatization makes use of a vocabulary and morphological analysis of the words to return the dictionary form of a word which is called the lemma [35].

There are different approaches to transforming normalized tokens into numerical representations. One of the most basic techniques for the numerical representation of data is one-hot encoding. One-hot encoding maps each token to a unique integer index i and stores this representation in a binary vector. The vector is of length N (the fixed size of the vocabulary), and all entries are zeros except at index i, which is 1. However, it creates a sparse high-dimensional representation of the document. In the BOW model, a document is represented as a set of tokens along with their associated term frequency in the document [4]. We denote the term frequency $tf_{t,d}$ to be the number of occurrences of term t in document d. The BOW representation is easy to interpret and adds information about the frequency (instead of a binary encoding), however, the frequency might be relative to the collection of documents. In other words, there might be terms that occur too often in the document collection to be meaningful to determine relevant similarity. Term Frequency-Inverse Document Frequency (TF-IDF) addresses this problem by measuring the statistical significance of words in a set of documents. Thereby, it re-weights $tf_{t,d}$ by the inverse document frequency df_t that describes the number of documents that contain the term t (further information about the formula can be found in [35]). Finally, we introduce word embeddings, such as word2vec[36] or GloVe [42]. In contrast to the sparsity of one-hot word vectors, word embeddings are dense floating-point vectors that are learned from data. The word2vec algorithm, for example, learns to represent words in a continuous vector space where semantically similar words are closer to each other. One can simply rely on pre-trained word embeddings or create word embeddings specifically for a set of documents. Although word embeddings are dense vectors, they normally require fewer dimensions than one-hot encoded representations and are able to indicate semantic relationships between words [12]. In the process of feature selection,

one determines the most relevant features to the classification task to improve the performance of the model [4].

2.2.2 Logistic Regression

We will first derive the formula for binomial logistic regression and later introduce the multinomial extension for using multiple labels. The latter is the baseline model we aim for.

Despite its name, logistic regression is a linear model for classification that optimizes the likelihood of the given observations. First, we look at the binary case of having two labels, i.e. the target variable $y_i \in 0, 1$ for a data point i. Let X_i be the feature vector, e.g. normalized BOW vector, w the weights and w_0 the bias to be found. A basic linear predictor is $p_i = X_i w + w_0$ as in linear regression [4]. Instead of using p_i to directly fit the true label y_i, we look at the probability of observing $y_i = 1$:

$$p(y_i = 1|X_i) = \frac{\exp(X_i w + w_0)}{1 + \exp(X_i w + w_0)}.$$

This function is called a sigmoid or logistic function. Applying a logit transformation on the above, we obtain the log odds

$$\log \frac{p(y_i = 1|X_i)}{1 - p(y_i = 1|X_i)} = X_i w + w_0,$$

that can be modeled by a linear combination of the features. Thus, the decision boundary of the logistic regression is linear and the class label with the highest probability will be assigned to the data point.

The classifier is trained by selecting the weights and biases that maximize the log-likelihood for all the observations. A common problem with training an ML model is that the model is easily overfitted to the training data and does not perform well on unseen data [8]. To avoid this, one can introduce a regularization term $r(w)$ that penalizes high-valued weights. One can adjust the effect of regularization by defining C as the inverse of the regularization strength, where a smaller C means stronger regularization. As a result, logistic regression with regularization minimizes the following cost function:

$$\min_w C \sum_{i=1}^{n} (-y_i \log(p(y_i = 1|X_i)) - (1 - y_i) \log(1 - p(y_i = 1|X_i))) + r(w).$$

The most frequently used penalties are L1 regularization $r(w) = \|w\|_1$, L2 regularization $r(w) = \|w\|_2^2/2$ or the combination of both defined by ElasticNet $r(w) = (1-\rho)w^T w/2 + \rho\|w\|_1$ where ρ controls the strength of L1 regularization compared to L2 regularization. There are several approaches to solve the above optimization problem, such as stochastic gradient descent, coordinate descent, or quasi-Newton methods (more information about numerical optimizers for logistic regression can be found here [38]). After solving the optimization problem for the given observations, we obtain the fitted model weights that we can use to predict the probability of the positive class:

$$p(X_i) = \sigma(X_i w + w_0) = \frac{1}{1 + e^{-X_i w - w_0}}, \tag{2.1}$$

where σ is the sigmoid function.

The binary case can be extended to having multiple classes [1]. Let $y_i \in 1, ..., K$ be the encoded target variable for the data point i. In contrast to the weight vector earlier, we now have a matrix of weights W, where each row vector W_k belongs to a class k. Thus, the probability of observing $y_i = k$ is defined as:

$$p(y_i = k|X_i) = \frac{\exp(X_i W_k + W_{0,k})}{\sum_{l=0}^{K-1} \exp(X_i W_l + W_{0,k})}.$$

Adding a regularization term $r(W)$, the optimization problem can be described as

$$\min_W -C \sum_{i=1}^{n} \sum_{k=0}^{K-1} [y_i = k] \log\left(p(y_i = k|X_i)\right) + r(W),$$

where $[y_i = k]$ yields 0 if the expression is false and otherwise 1.

To use logistic regression for multi-label classification, one can apply problem transformation techniques to convert multi-label data to binary data [43]. One of them is binary relevance that creates a single binary classifier for each different label of the original data. Thus, it has linear complexity with respect to the number of labels but does not take into account label correlations. Label powerset addresses this problem by creating one label for each subset of labels. Since the possible powerset combinations are 2^L for L distinct labels in the data, it is recommended for data sets with a small number of labels. Classifier chains are another approach that creates $|L|$ binary classifiers as in binary relevance. These binary classifiers are linked

along a chain $C_1, ..., C_{|L|}$ where each classifier deals with the binary relevance problem associated with label $l_j \in L$. Each C_j is responsible for learning and predicting label l_j, complemented by the prior binary relevance predictions in the chain $l_1, ..., l_{j-1}$. The library of scikit-learn offers two multi-label models for logistic regression: MultiOutputClassifier and ClassifierChain [2]. MultiOutputClassifier is based on binary relevance and fits one classifier per target.

2.3 Deep Learning in Text Classification

A special sub-field of ML is DL that puts an emphasis on learning successive layers of meaningful representations of the data. As the depth of a model, i.e. the number of layers contributing to the model, can be scaled up with the computational resources available, deep neural networks are particularly useful for tasks in NLP. The abundance of data and the complex dependencies in natural language are challenging to model with traditional ML-based models as described above. In this section, we will first describe deep neural networks by introducing Feedforward Neural Network (FNN) and then we will move our focus to sequential modeling and outline RNN and Transformer.

2.3.1 Feedforward Neural Network

A Feedforward Neural Network (FNN) or also called a multi-layer perceptron is composed of layers of interconnected neurons, each of which performs basic mathematical operations. The term neural network is a reference to neurobiology and some of the concepts in DL stem in part from our understanding of how the brain works. The goal of FNNs is to approximate some function f. In the case of classification, $y = f(x)$ maps an input x to the class y. It thereby defines a mapping $y = f(x, w)$ and learns the value of the parameters w that result in the best function approximation given the labeled data $(x^{(i)}, y^{(i)})$ [15].

Consider the simplest possible neural network that only consists of a single neuron. This neuron is a computational unit that takes as input $x = (x_1, ..., x_n)$ and outputs

$$f(x, w) = a\left(\sum_{i=1}^{n} w_i x_i + b\right),$$

where a is called the activation function and b is the bias or intercept term.

Among other activation functions like hyperbolic tangent, tanh, or rectified linear function, a typical choice is the sigmoid function $a(z) = (1 + \exp(-z))^{-1}$. Using a single neuron with a sigmoid activation function thus corresponds to the input-output mapping of logistic regression. FNNs, which consist of several layers, consequently use a composition of many different functions. For example, we model a network with k layers, where $f^{(1)}$ describes the input layer, $f^{(i)}$ for $i \in 2, ...(l-1)$ the hidden layers and finally $f^{(l)}$ the output layer. The model now composes these functions in a chain

$$f(x) = f^{(k)} \circ f^{(k-1)} \circ ... \circ f^{(2)} \circ f^{(1)}(x).$$

The term "feedforward" is used to describe models in which information flows in a single direction, starting from x, passing through the intermediate computations used to define f, and ending with the output y. Unlike RNNs, there are no feedback connections in which the model's outputs are fed back into itself.

2.3.2 Recurrent Neural Network

The problem of FNNs is that they have no memory and are not able to process sequential data efficiently. One workaround would be to turn a sentence into a single data point that is shown to the network at once. However, RNN is a neural network that is specialized for processing a sequence of values $x^{(1)}, ..., x^{(\tau)}$ where τ can even vary for most RNNs. Similar to how we humans read a sentence, the model is able to process information by iterating through the sequential input while maintaining a state that contains information from the past inputs. When you read this sentence, for example, we process it word by word while keeping in mind what came before [12].

RNNs take into account the sequential nature of the input by computing a set of feedback weights in a hidden state vector $h^{(t)} = f(h^{(t-1)}, x^{(t)}, w)$ for the time step t. During training, the model learns to use $h^{(t)}$ as a lossy summary of the task-relevant aspects of the seen inputs. The RNN unfolds a recurrent or recursive computation into a computational graph with a repetitive structure. This results in parameter sharing across the time steps of the deep network structure. Parameter sharing allows us to extend and apply the RNN to a variety of examples. For example, consider the sentences "I moved to Sweden in 2020." and "In 2020, I moved to Sweden." that describe the same information. A FNN would have learned separate parameters for a fixed position in the sentence and it might process both sentences differently. However, a RNN shares the same weights across several time steps and is able

to learn representations disregarding the position in the sequence data.

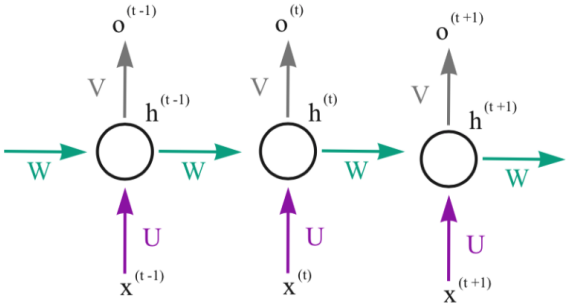


Figure 2.2: Unfolding of the recurrent computations in an RNN. For each input $x^{(t)}$, the hidden state $h^{(t)}$ and the output $o^{(t)}$ are computed. The parameters of this RNN are the three weight matrices U (mapping from input to hidden layer), V (mapping from the final hidden layer to output), W (mapping across hidden states).

Figure 2.2 describes the unfolded computations in an RNN. At time step t, the input data $x^{(t)}$ enters the network. First, the hidden state $h^{(t)}$ is computed by taking into account the former hidden state and the input. Then the hidden state is passed to computing the hidden state for the input at the next time step [23].

To train a neural network model, a loss function must be formulated for which the parameters are updated based on the given data. More information about the back-propagation algorithm can be found in section 10.2.2 in [15]. RNNs are prone to suffer from two types of problems in the training process. Since weights are propagated through time, they are multiplied recursively in the previous functions and the gradient can vanish or explode. Vanishing gradient occurs when the weights are small and thus the subsequent values get smaller till they vanish at some point. In contrast, in exploding gradient, the weights are large and thus the final value might approach infinity. As a result, the RNN architecture is very sensitive to the number of time steps, and useful gradient information from the end of the output is poorly transferred to the beginning of the input, resulting in a degradation of long-term dependencies.

In order to address this issue, a particular type of hidden layer known as Long Short-Term Memory (LSTM) is employed, and also Gated Rectified Unit (GRU), a variation of LSTM. Both types of networks are designed with internal mechanisms called gates, that control the information that passes through the network. During training, the forget gate determines which pieces of information should be retained and which should be discarded.

2.3.3 The Encoder-Decoder Architecture

In certain NLP tasks such as machine translation, the input as well as the output of the model are sequences of arbitrary length. Thus, the model should be able to process as well as output sequential data and this can be addressed by an encoder-decoder or sequence-to-sequence architecture. Figure 2.3

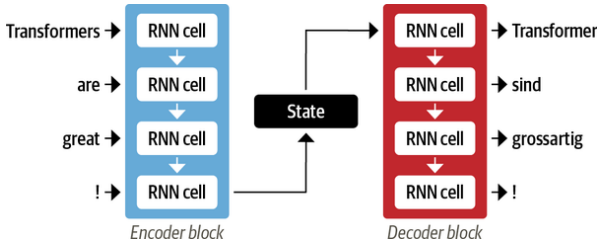


Figure 2.3: Illustration of an encoder-decoder architecture. Here, the encoder and decoder contain four RNN cells but there are typically more recurrent layers or there could be other neural network architectures [55].

illustrates the encoder-decoder architecture. The input sequence (in this case the sentence "Transformers are great!") enters the encoding block that encodes the information into a numerical representation, which is referred to as the last hidden state. The decoder then processes the last hidden state and generates the output sequence. One drawback of this architecture is the information bottleneck created by the final hidden state of the encoder. It is responsible for representing the information of the entire input sequence, as the decoder only has access to this when producing the output. This becomes particularly difficult when dealing with longer sequences, where important information from the beginning of the sequence might be lost. Another weakness of RNNs is that the computations are sequential and thus cannot be parallelized across the sequential input data.

2.3.4 Attention Mechanisms

One approach to resolve the information bottleneck caused by the last hidden state is to use a mechanism that is called attention. Instead of producing a single last hidden state for the sequential input, the encoder makes now all hidden states available. But since this could easily get inefficient, we need a mechanism that focuses on specific parts of the hidden states. This is where attention becomes important. This mechanism which tries to mimic the internal cognitive structure of our brains was a key aspect of the development of the transformer model. In 2017, the R&D department at Google proposed

the novel idea of using self-attention internally in the encoder and decoder in the famous "Attention is all you need" paper [56].

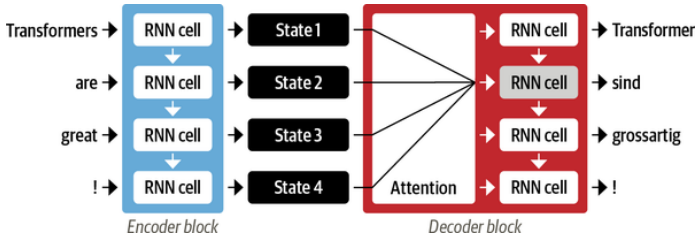


Figure 2.4: The attention mechanism in the encoder-decoder architecture illustrated for predicting the third token in the output. RNNs are used [55].

As pictured in Figure 2.4, the hidden states from the RNN cells are shared with the decoder block. Attention then assigns weights to each hidden state at every time step in the decoding. Thereby, it selects relevant information while fading out the rest.

2.4 Transformers

The development of transformers has revolutionized language processing. Their underlying architecture, including the mechanism of self-attention, provides a much faster and more parallelizable model than traditional RNN-based approaches. This is an ideal combination in NLP considering the volume of language data available. In 2017, Vaswani et al. [56] proposed the Transformer, which initiated the research and development of further transformer models. Figure 2.5 describes the releases of various transformers till 2021. Worth mentioning is the recent release of ChatGPT by OpenAI in 2022.

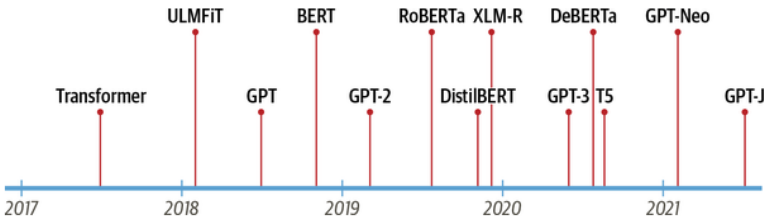


Figure 2.5: Timeline of releases of transformers in NLP from 2017 till 2021 [55].

In this section, we will look at transformers by first outlining transfer learning in NLP and then providing an understanding of the Transformer architecture. We conclude this section by listing several transformer models.

2.4.1 Transfer Learning

Initially, NLP models were trained by starting with a random initialization of the weights and then training the model for each downstream task [34]. However, transfer learning has emerged as a useful technique, in which a neural network is first pre-trained on a general task and then fine-tuned for a specific task. This approach has enabled deep learning models to achieve faster convergence and relatively less data for fine-tuning. The idea of transfer learning stems from the field of computer vision where convolutional neural networks were pre-trained on large-scale data sets such as the ImageNet data set [17].

The first step of transfer learning is pre-training, which is mainly based on self-supervised learning on unlabeled data [47]. The model is trained on data-rich tasks that ideally lead to general-purpose knowledge that allows the model to "understand" text. This learned knowledge can then be transferred to downstream tasks where the model is trained on a supervised task with labeled data. For effective transfer learning, the fine-tuning data set should be related to the pre-training data set. Architecturally, this process entails dividing the model into two parts—a body and a head. The weights of the body are trained during pre-training which is generally referred to as semi-supervised learning resulting in general features of the source domain. The head represents the task-specific network that is used for supervised training on a labeled training data set. This results in the dependence of supervised fine-tuning on unsupervised language modeling [34].

The first novel approaches to effectively implement transfer learning in NLP started in 2017 and 2018. Researchers at OpenAI achieved impressive results on a sentiment classification task by leveraging features extracted from unsupervised pre-training [45]. After these insights, Universal Language Model Fine-tuning (ULMFiT) was developed and introduced a comprehensive framework that enabled the adaptation of pre-trained LSTM models for diverse tasks. ULMFiT can be divided into three main steps: pre-training, domain adaption, and fine-tuning [55]. Here, the model is first pre-trained on data that is based on a wide range of already existing texts from sources such as Wikipedia. This process is also followed by language modeling, but the training data is adapted to the in-domain corpus. Finally, the language model

is fine-tuned by adding a classification layer for the downstream task.

2.4.2 Architecture

The original Transformer [56] is based on an encoder-decoder architecture, similar to what we have seen in Section 2.3.3. However, there are many variations that can roughly be divided into three types: encoder-only, decoder-only, and encoder-decoder models.

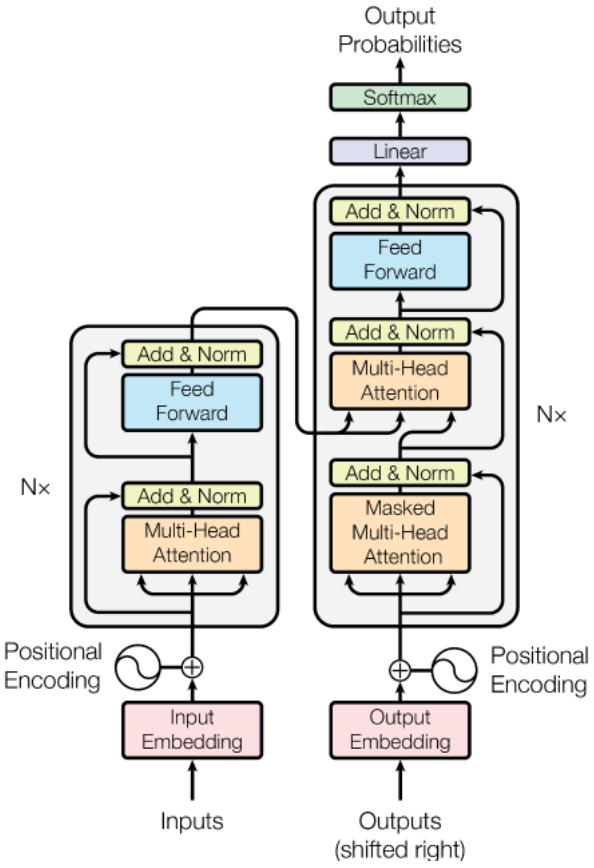


Figure 2.6: Encoder-decoder architecture of a transformer. The left side describes one of the N encoding layers and the right side one of the N decoding layers [56].

We will briefly walk through the components of the encoder-decoder architecture illustrated in Figure 2.6. Starting with the encoder, the sequential input data is first converted into token embeddings. Then, positional embeddings are created that provide information about the position of each token. As the attention mechanism lacks knowledge about the relative positions of tokens, it

is important to include positional information in the input to effectively model the sequential nature of the input. In addition to that, token embeddings are projected into the three vectors *query* Q, *key* K and *value* V. Furthermore, the encoder consists of a stack of encoder layers to be able to capture complex representations. In [56], they used $N = 6$ identical layers and each has two sub-layers. The first sub-layer is a multi-head self-attention layer and the second is a fully connected FNN that allows parallelization. Furthermore, one can find a "Add & Norm" module after each sub-layer of the encoder and also decoder. This module incorporates residual learning and layer normalization to the respective output. Residual learning is a training technique for deep neural networks that adds links from the input to the output of the sub-layer. To summarize, the main idea of the encoder is to modify the input embeddings to create representations that add contextual information to the sequence.

Let us delve deeper into (multi-headed) self-attention. As we have seen in Section 2.3.4, attention allows networks to focus on parts of the sequence by computing weights of each encoder's hidden state at a given decoding time step. In self-attention, however, these weights are computed for all hidden states of a stack. In case of the encoder, the transformer can access information from any part of the input sequence to compute a weighted average of each embedding. To calculate the attention weights, an attention function needs to be formulated. Among other approaches to computing the attention score, scaled dot-product attention as proposed in [56] is defined as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.2}$$

where d_k is the dimension of the queries and keys. The concept of self-attention can be extended to a multi-headed version. This means, that the vectors Q, K, V are now projected in h dimensions, where h is called the number of heads. Each projection carries its own set of learnable parameters that allows focusing on several aspects of similarity. Vaswani et al. [56] chose $h = 8$ parallel attention layers.

In summary, the encoder receives the embedded input sequences, encodes them and passes its output to each decoder layer. Then, the decoder takes this as input and predicts the most probable next token in the sequence. The chosen token of this step is then fed back into the decoder to generate the next token. Thus, the decoder's input is a combination of the encoder's output and its own generated output. This process is repeated until a special end-of-sequence token is reached.

The decoder component looks similar but slightly different from the

encoder component. The decoder layers also contain a self-attention layer and a feed-forward network but it has an additional encoder-decoder attention layer in between. This layer is needed to combine tokens from two different sequences (the encoders' output and its own output). Furthermore, it is also necessary to ensure that the decoder does not cheat during training by simply copying the given solution. This is done by the masked multi-head self-attention layer which sets future position values to $-inf$ before applying softmax in Equation (2.2) ensuring that the decoder generates tokens only based on the past outputs. After the final decoder layer, a linear layer maps the output to the vocabulary yielding non-normalized probabilities, also called logits. Finally, the softmax layer normalizes the logits resulting in output probabilities. After the task-independent body of the transformer, there is usually a task-specific head [55].

In the next subsections, we will describe a selection of encoder-only transformers. GPT-3 [9] is an example of a decoder-only transformer that does not require fine-tuning at all. The resource-intensive nature of the model, however, makes pre-training expensive and since GPT-3 is an auto-regressive model that learned to predict the next token in a sequence it is mostly used for text generation tasks [55].

2.4.3 BERT

In 2019, researchers at Google AI Language [18] released Bidirectional Encoder Representations from Transformers (BERT). It leverages the encoders in a transformer as a sub-structure to pre-train models for various NLP tasks such as text classification, question answering, or text summarization. BERT is a multi-layer bidirectional Transformer encoder as described in Section 2.3.3. BERT utilizes bidirectional self-attention meaning that the input is processed from right-to-left and left-to-right allowing self-attention to attend context from both the right and left side of the masked token.

BERT's input is illustrated in Figure 2.7. To get the token embeddings, they utilize WordPiece embeddings with a vocabulary size of 30,000. Each input sequence starts thereby with the special classification token [CLS]. The input can either be a single sentence or a pair of sentences. For sentence pairs, the sentences are joined with the special token [SEP] into a single sequence. Furthermore, the segment embedding indicates which token belongs to which sentence by marking them as *A* or *B*. Finally, the positional embedding describes the absolute position of a token in the sequence. Consequently, the model is limited to an input maximum of 512 tokens.

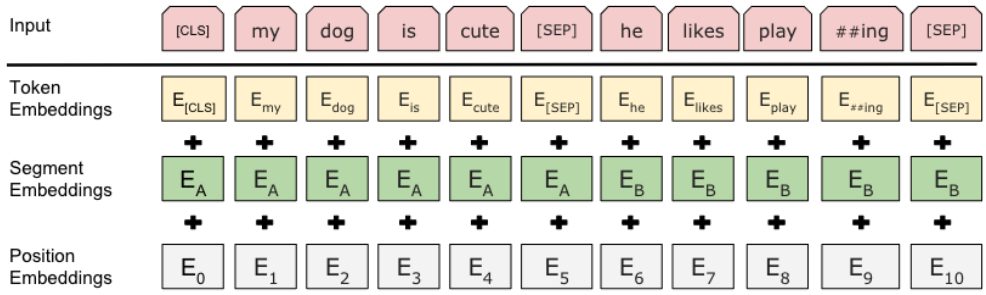


Figure 2.7: Illustration of BERT's input [18].

In the original paper, Devlin et al. [18] introduced two model configurations that are summarised in Table 2.1.

Model size	N	H	A	#Parameters
BASE	12	768	12	110M
LARGE	24	1024	16	340M

Table 2.1: BERT models. The number of layers (i.e., Transformer blocks) is denoted as N, the hidden size as H, and the number of self-attention heads as A.

The transfer learning framework consists of two steps: pre-training for language understanding and then fine-tuning for the specific task. Language understanding is achieved through training using the mechanisms Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). In MLM, some random words in the input data are masked and the learning objective is to predict the original vocabulary id of the masked word. NSP is used to learn to understand the relationship between two sentences. Given two sentences *A* and *B*, the task is to decide whether *B* is truly the next sentence following *A* or not. This testing is done by replacing the sentences in 50% of the cases with random sentences from the corpus. The data used for pre-training consists of BooksCorpus (800M words) and English Wikipedia (2,500M words).

To utilize BERT for a downstream task, it can be fine-tuned with one additional output layer. This is a supervised downstream task requiring labeled data.

2.4.4 DistillBERT

In 2020, Sanh et al. [49] introduced DistilBERT, a modified and optimized version of the original BERT model. It uses knowledge distillation to shrink

the size of BERT by 40% while keeping 97% of its language understanding capabilities and being 60% faster.

One of the primary modifications in DistilBERT is the use of knowledge distillation. DistilBERT is trained using a large-scale BERT model as a teacher. Instead of relying solely on ground truth labels, DistilBERT learns from the teacher's soft targets, such as logits or probabilities. This process allows the smaller model to acquire knowledge from the teacher model and compress the information into a more compact representation. To achieve a reduced model size, DistilBERT eliminates token type embeddings and pooler layers, which are not crucial for many downstream tasks. Additionally, the number of layers in DistilBERT is reduced by a factor of two. Another modification in DistilBERT involves the pruning of attention heads. Attention heads in BERT capture different contextual relationships between words, but not all of them are equally important. DistilBERT applies a pruning technique to remove the least important attention heads. The remaining attention heads are fine-tuned to maintain performance, resulting in a more compact model.

2.5 Evaluation

To be able to compare classification models, evaluation metrics are required. In the following, we will focus on two different perspectives of measuring the quality of a model. First, we will derive a variety of metrics that quantify how well a classification model is solving a classification task given the true labels. Second, we introduce some cost indicators of using a ML-based model.

2.5.1 Performance

There exists a variety of performance metrics that try to capture how well a model is able to predict the target. A common presentation of the results of a supervised learning task is the confusion matrix.

		PREDICTION			
ACTUAL	Classes	a	b	c	d
	a	TN	FP	TN	TN
	b	FN	TP	FN	FN
	c	TN	FP	TN	TN
	d	TN	FP	TN	TN

Table 2.2: Confusion matrix for the multi-class case where b is the reference class. It contains the values TP [=positive instances that were correctly classified as positive], TN [=negative instances that were correctly classified as negative], FP [negative instances that were incorrectly classified as positive] and FN positive instances that were incorrectly classified as negative].

Table 2.2 describes the results of a multi-class classification by comparing the model prediction to its real value. The confusion matrix can be expanded by adding a row and a column for each additional class. To compute True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) for each class, one class is selected as positive and the other classes are treated as negative. The importance of these four values can vary depending on the classification setting. There are several performance metrics that make use of the values provided by the confusion matrix. We will first look at the binary case and then continue to multi-class metrics. Accuracy describes the percentage of correct predictions over all predictions [27]:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FN + FP}.$$

Recall instead computes the percentage of positive instances the model was able to predict correctly. Precision calculates the percentage of how many positive predictions are indeed true. Both metrics are computed in the following:

$$\text{Recall} = \frac{TP}{TP + FN}$$
$$\text{Precision} = \frac{TP}{TP + FP}.$$

A popular aggregated evaluation metric is the F_β-score that combines the two metrics recall and precision. The parameter β can be adjusted depending on the importance of each metric but a commonly used value is $\beta = 1$ leading

to the harmonic mean between recall and precision values. F1-score reaches its best value at 1 and worst at 0.

$$F_\beta = \frac{(1+\beta^2)(\text{Precision} \cdot \text{Recall})}{\beta^2 \cdot \text{Precision} + \text{Recall}}$$
$$F_1 = \frac{2TP}{2TP + FP + FN}$$

Receiver Operating Characteristics (ROC) curves evaluate the performance of the classifier graphically by plotting the true positive rate against the false positive rate.

$$TPR = \frac{TP}{TP + FN}$$
$$FPR = \frac{FP}{FP + TN}$$

Class imbalances may cause the ROC curve to not properly represent the performance. Area under ROC Curve (AUC) aggregates the information by ROC curve by calculating the entire area underneath the curve:

$$AUC = \int_{-\infty}^{\infty} TPR(T)FPR'(T)\,dT.$$

It is independent of the chosen threshold and invariant to a priori class probabilities.

The above metrics can also be used in the case of multi-label classification. Precision at k ($P@k$) would be defined as the number of correct predictions considering only the top k labels (ranked by prediction score) divided by k. Common values for k are $\{1, 3, 5\}$ [53]. Let us look at an example. For the ground truth multi-labels $y = (2, 5, 10, 11)$ and the prediction $p = (2, 4, 5, 13)$ ranked by the prediction confidence, we get $P@1 = 1$, $P@2 = 0.5$, $P@3 = 0.66$. Similarly, one can compute recall at k. One can also compute the F1-score by using an aggregation approach. More specifically, one can see a multi-class or multi-label classification problem with $|C|$ classes as having $|C|$ binary classifiers. Thus, a metric score can be computed for each binary classifier and subsequently aggregated using various approaches, such as taking the sample, micro or macro average. To compute the macro F1-score, one first calculates $Precision_k$ and $Recall_k$ for each class k and then takes the

arithmetic mean:

$$\text{MacroPrecision} = \frac{\sum_{k=1}^{K} \text{Precision}_k}{K}$$
$$\text{MacroRecall} = \frac{\sum_{k=1}^{K} \text{Recall}_k}{K}$$
$$\text{MacroF1} = 2 \cdot \frac{\text{MacroPrecision} \cdot \text{MacroRecall}}{\text{MacroPrecision}^{-1} + \text{MacroRecall}^{-1}}.$$

Thus, each class has the same weight in the average disregarding class frequencies. To address this issue, one can compute a weighted average that weights the F1-score of each class by the number of samples from that class. For micro averaged F1-score, we calculate metrics globally by counting the total TP, FN and FP. As a result, micro-averaged precision and recall yield the same value that is set to be the micro-averaged F1-score.

LRAP is another metric based on the notion of label ranking and is similar to $P@k$. For each ground truth label, it indicates what fraction of the labels that were ranked higher are actually true labels. LRAP ranges from 0 to 1, which is the best value. Given a binary indicator matrix of the ground truth labels $y \in \{0,1\}^{n_{\text{samples}} \times n_{\text{labels}}}$ and prediction score associated with each label $\hat{f} \in \mathbb{R}^{n_{\text{samples}} \times n_{\text{labels}}}$, LRAP is defined as:

$$LRAP(y, \hat{f}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \frac{1}{\|y_i\|_0} \sum_{j:y_{ij}=1} \frac{|\mathcal{L}_{ij}|}{\text{rank}_{ij}},$$

where $\mathcal{L}_{ij} = \{k : y_{ik} = 1, \hat{f}_{ik} \geq \hat{f}_{ij}\}$, $\text{rank}_{ij} = |\{k : \hat{f}_{ik} \geq \hat{f}_{ij}\}|$.$|\cdot|$ denotes the cardinality of the set (# elements), and $\|\cdot\|_0$ is the ℓ_0 norm (#nonzero elements in a vector) [3].

Given a hierarchical taxonomy for the labels, standard metrics like precision, recall and F1-score can be adapted [20]. Therefore, one compares the set of the ancestors labels of the prediction and the ground truth. Let $X = \{\hat{f}_i | 1, ..., M\}$ be the set of the predicted labels and $Y = \{y_i | 1, ..., N\}$ the set of the true labels. We define an augmentation function $An_j(x)$ that returns the ancestors of the node x up to the j^{th} level. Thus, we can represent the set of ancestors of X and Y to the j^{th} level as $X_{aug} = X \cup \{An_j(x_i) | 1, ..., M\}$ and $Y_{aug} = Y \cup \{An_j(y_i) | 1, ..., N\}$, respectively. Subsequently, standard metrics can be applied on those augmented sets.

2.5.2 Usage Cost

In contrast to the previously described metrics, we disregard model predictions for now and focus on the model architecture to compare complexity and efficiency. Therefore, we will describe multiple cost indicators. However, there can be several factors that influence cost indicators. Such as the hardware, the model runs on (e.g., Central Processing Unit (CPU), GPU, or TPU) or the model's framework (e.g., JAX, PyTorch, or TensorFlow), or even the programming skills [16]. The number of trainable parameters of a model is an indirect indicator of computational complexity, especially used in the NLP domain [16]. It also provides an estimate of memory usage (during inference).

Another informative indicator of efficiency is the speed of the model. The hardware has a strong influence on the speed and therefore the same hardware or normalization is required to achieve meaningful and fair comparisons [13, 16]. One can report speed in different forms. A selection can be found here:

- *Throughput* describes the number of examples (e.g. tokens) that are processed during a specific time period (e.g. tokens per second).
- *Runtime/ Wall-clock time* measures the time it takes the model to process a fixed set of examples. One can measure the total training time up to convergence.
- *Latency* most often refers to the model inference time given one or a batch of examples. The usual representation of latency is "seconds per forward pass", where a forward pass describes the process in which input features enter a neural network, pass through the layers, and are modified for the final prediction. In contrast to throughput, latency takes into account parallelism and thus is a better representation for real-time systems that require user input.
- *Memory Access Cost* describes the number of memory accesses. It usually accounts for a large part of the runtime and is considered the real bottleneck when operating on modern platforms with high computational power like GPUs and TPUs.

A commonly used metric for the computational cost of a model is FLOPs, the absolute number of floating point multiplication-and-addition operations [16]. Note that the other metric FLOPS describes the number of floating point operations per second. FLOPs are mainly calculated using theoretical values. In 2020, a research team of OpenAI proposed an estimation of

computing FLOPs for neural language models [25]. First, we define several model-specific parameters: n_{layer} (number of layers), d_{model} (dimension of the residual stream), d_{ff} (dimension of the intermediate feed-forward layer), d_{attn} (dimension of the attention output), and n_{heads} (number of attention heads per layer). In order to achieve much cleaner scaling laws, embedding parameters and sub-leading terms such as nonlinearities, biases, and layer normalization are omitted. The model size is then interpreted as the number of non-embedding parameters

$$N \approx 2d_{model}n_{layer}(2d_{attn} + d_{ff})$$

Consequently, the number of multiplication-and-addition operations in a forward pass with n_{ctx} tokens in the input context is estimated by:

$$C_{forward} \approx 2N + 2n_{layer}n_{ctx}d_{attn}$$

A backward pass is approximately twice as computationally expensive as the forward pass and thus there are $C \approx 6N$ FLOPs per training token disregarding embedding. However, FLOPs does not take into account information such as the degree of parallelism (e.g., depth, recurrence) or hardware-related details like the cost of memory access.

Since cost indicators are mainly used to compare two or more models or methods, one might have the possibility to keep specific cost indicators fixed for the models to get a fair comparison. Dehghani et al. [16] outline two main approaches, namely parameter-matched comparisons—models with a similar number of trainable parameters—and flop/computation matched comparisons—models with a similar computational budget, such as similar FLOPs. Depending on the context of use, some cost indicators may be more important than others. Applications that rely on the user experience might require a good latency, for instance. Other models, such as certain recommender systems, need to be retrained more frequently in the face of new data, and thus training time may be more relevant.

Chapter 3

Related Work

This chapter provides an overview of the state-of-the-art technologies and approaches in hierarchical multi-label text classification and comparison of transformers.

3.1 Hierarchical Multi-Label Text Classification

Research on HMTC focuses in particular on the challenges of how to model dependencies among the hierarchical labels, the correlations between label and text, and how to evaluate these models.

3.1.1 Proposed Approaches

Most related work on hierarchical text classification can be roughly divided into local and global approaches, depending on how they deal with the label hierarchy. Local approaches construct multiple classifiers for each node or level, while global approaches create only one classifier for the whole graph. To mention one local approach, Costa et al. [14] employed a top-down hierarchical classification approach using decision-tree-based classifiers. One global approach could be to flatten the hierarchy, thereby ignoring vertical dependencies and redefining the hierarchical classification problem as a multi-label classification problem. Also, there exist hybrid models that combine global and local optimization, such as in the research of Wehrmann et al. [61]. They propose a unified framework that combines the global output of the network and the local outputs of each hierarchical category level.

Deep learning architectures mainly follow global approaches [65]. There are hierarchy-based approaches that are based on employing two encoders, one for encoding the class hierarchy and one for the textual input. Both representations can be learned separately and then combined. In 2020, Zhou et al. [65] introduced two variants of a hierarchy-aware global model. Alongside another deductive variant, we will briefly describe the multi-label attention variant called HiAGM-LA. Therefore, they use a traditional text encoder and a hierarchy-aware structure encoder—for example a Tree-LSTM or a graph convolutional neural network—for modeling hierarchical label relations. To aggregate the information of both encoders, HiAGM-LA extracts inductive label-wise text features for which it conducts multi-label attention.

Instead of modeling the encoding of text and label hierarchy separately, Wang et al. [60] introduced Hierarchy-guided Contrastive Learning (HGCLR) in 2022. The model migrates the hierarchy information into the BERT encoding through contrastive learning using a graph encoder. Their methodology was based in part on the research of Zhou et al. [65]. Jiang et al. [24] proposed Hierarchy-guided BERT with Global and Local Hierarchies (HBGL), a framework that does not rely on helper modules like graph encoders. It makes use of the hierarchical information by modeling the hierarchy and semantic information directly with BERT. First, HBGL learns the label embeddings from the global hierarchy by applying an attention mask on the adjacency matrix of the taxonomy. It then learns to predict the document labels one by one in the order of the local hierarchy (from the root to the most specific label). Their research yielded an improvement to HGCLR, although it requires additional iterations to predict labels. HBGL first needs to predict upper-level labels and can then continue predicting the current level. Liang et al. [31] utilize BERT and map the learned latent features to hierarchical labels with a delicate hierarchy-based loss layer.

In 2022, Lu et al. [32] proposed Multi-task Hierarchical Cross-Attention Network (MHCAN). The cross-attention mechanism incorporates hierarchical label embedding and text representation to capture the dependencies between levels. They experimented with several modifications, such as introducing domain-adaptive pretraining and adversarial training. In addition to that, they combined the predictions of the experiments resulting in an ensemble learning method of a voted model that improved the generalization ability of the classifiers.

Research by Galke et al. [22] in 2023 investigated the performance of graph-, sequence- and hierarchy-based models for multi-label text classification. Their results concluded that the additional use of graph-based methods

to exploit the hierarchical taxonomy for multi-label classification does not outperform transformer models. They recommend using a sequence-based language model when there are sufficient computational resources available otherwise rely on WideMLP (one wide hidden layer with 1,024 rectified linear units). WideMLP is a BOW-based multi-layer perceptron with TF-IDF.

3.1.2 Data

There exist several publicly available data sets that include a hierarchy label taxonomy. Table 3.1 describes a selection of data sets that are used in various research.

Data set	Depth	#Classes for each level/ total	#Instances	Multi-path?	Used by
RCV1-V2 [30]	4	103	804,414	Y	[11, 22, 60]
Web-of-Science [26]	2	141	46,985	N	[11, 60]
NYTimes [48]	8	166	36,471	Y	[60]
SharedTask5 [28]	3	21/ 260/ 1,272	100,000	?	[32]
EURLEX-57K [10]	5	4271	57,000	?	[11, 44, 60]

Table 3.1: Public data sets used in research concerning HMTC. The column Multi-path indicates whether a data point can belong to several labels from the same level. Furthermore, we list some studies that used these data sets.

3.1.3 Evaluation

Due to the hierarchical dependencies among the labels in HMTC, the evaluation of the performance of the model can be challenging. Frequently used metrics in HMTC are based on modifications of standard metric, such as Recall@5 [11], sample-based F1 [22], Macro-F1 and Micro-F1 [11, 22, 60]. Flat and standard metrics such as precision, recall, and F1-score do not take into account the similarity of labels and treat mispredictions equally. Labels that belong to the same node parent are more similar and mispredictions could be penalized less than mispredictions for a completely

different branch, for instance. Falis et al. [20] propose count-preserving hierarchical evaluation metrics that include hierarchical standard metrics such as hierarchical precision.

Since HMTC are prone to suffer from a long-tailed distribution of labels, the evaluation needs to take into account the label imbalance. There have been approaches to address this problem by introducing balancing methods. Lu et al. [32] re-weight the loss function, in their case binary cross-entropy, by the effective number of samples for each class. In addition to that, the labels are weighted differently with respect to their classification level before computing the total loss. The labels of the first level are weighted with 0.05, whereas the labels of the second and third levels are weighted with 1 and 5, respectively.

3.2 Comparative Studies on Transformers

The following section provides an overview of research concerning the comparison of transformers in text classification. As stated above, recent research by Galke et al. [22] suggested that sequence-based transformers still achieve better results despite the ignorance of class hierarchies. Thus, we will focus on the comparison of sequence-based models in multi-label and partly multi-class text classification.

3.2.1 Proposed Approaches

In 2019, Wang et al. [59] introduced the General Language Understanding Evaluation (GLUE) benchmark to evaluate the performance of models across a diverse range of existing natural language understanding tasks. There also exists a modified version SuperGLUE that contains a new set of more difficult language understanding tasks [58]. The leaderboard of the GLUE benchmark has become the key battleground to compare the performance of BERT, among other language models. Although the GLUE tasks possess distinct properties, they share a common objective of classification with the exception of one regression task [22]. However, multi-label text classification is not present in the leader board of GLUE, nor in recent CLEF or SemEval conferences [21].

The aforementioned research of Galke et al. [22] from 2023 provides a thorough comparison of the performance of sequence-based classifiers in multi-label classification. They state that the state-of-the-art in inductive text classification remains to fine-tune pre-trained transformer-based language models. Apart from BOW and graph-based models, they compare BERT (base

and large), DistilBERT, RoBERTa and DeBERTa, among other models. They state that transformer models are sensitive to the learning rate and thus they repeat experiments with (most importantly) lower learning rates during fine-tuning. The results of BERT-large are only marginally better than BERT-base for four out of five data sets. Their experiments conclude that DeBERTa sets state-of-the-art for single-label and multi-label text classification. Research of Fallah et al. [21] comes to the same conclusions regarding the overall outperformance of transformers and especially DeBERTa. They proposed architectures for the classification layers on top of transformers to get better performance in multi-label classification. Apart from introducing threshold selection methods, they compare variants of BERT (uncased-base versions) with a similar selection as in [22] but also DocBERT, a fine-tuned version of BERT (base and large) for document classification. The basic versions of BERT achieve better results than the basic version of DocBERT, which, however, is less trained in the fine-tuning process.

Arslan et al. [6] compared Pre-trained Language Models (PLMs) for multi-class text classification in the financial domain. They investigated BERT, DistilBERT, RoBERTa, XLNet (XLNet), and XLM (XLM) by using the corresponding HuggingFace Transformers library. They also tested FinBERT (FinBERT) a specialized pre-trained model for financial documents which, however, did not lead to improvements compared to the other generic BERT models. They assume that the financial text data to which the model is fine-tuned may differ significantly from the documents contained in the financial data sets they use. Overall, RoBERTa performs best on the data sets. Similar to Arslan et al., Wahba et al. [57] compared the above PLMs, excluding XLNet, with Support Vector Machine (SVM), a linear classifier. They used the LinearSVC algorithm from the Scikit-learn library to employ a one-versus-all multi-class strategy. This included preprocessing the data by lemmatization and word vectorization using TF-IDF for tri-grams. Shaheen et al. [51] studied the performance of transformer-based models combined with multiple strategies such as generative pretraining, gradual unfreezing, and discriminative learning rates. They tested BERT, DistilBERT, RoBERTa, XLNet and M-BERT.

Extreme Multi-label Text Classification (XMTC) is a sub-field of multi-label classification that deals with data sets consisting of thousands of labels. For XMTC, Qarei et al. [44] compared the performance of linear classifiers using the BOW representation to state-of-the-art DL approaches. They concluded that classical models such as support vector machines and logistic regression can compete with the DL-based approaches AttentionXML and

XML-CNN. However, one has to consider the nature of the XMTC task. In contrast to sentiment analysis or other NLP tasks that rely on strong semantic and contextual information, XMTC often deals with finding appropriate keywords for a news story on the BBC or an article on Wikipedia, for example. Thus, DL approaches cannot leverage their strong capabilities in semantic and contextual information extraction. Furthermore, there is often data scarcity where a large fraction of labels has very few training instances. Ragesh et al. [46] analyzed graph convolution networks with logistic regression as a baseline model, among others. They showed that logistic regression outperformed the advanced graph-based TextGCN method. They applied logistic regression on TF-IDF transformed word vectors with l2-regularization and hyper-tuned C over $[1e{-}5, 1e5]$ in powers of 10. Morales-Hernandez et al. [39] analyzed problem transformation techniques to convert multi-label data to binary data for logistic regression, among other linear classifiers. Therefore, they tested binary relevance, label powerset, and classifier chains on data sets with 17 labels. Overall, the label powerset performed the best, while binary relevance performed the worst by a greater margin.

All the above-mentioned research on multi-label or multi-class text classification mainly focuses on the performance of the models and not on the efficiency or usage cost. Galke et al. [22] provide a summary of the number of parameters for selected methods used in their experiments. However, we will discuss some research that takes into account the usage cost of transformers and in particular by computing FLOPs. Dehghani et al. [16] provide a comprehensive overview of cost indicators for measuring model efficiency in ML. They discuss the advantages and disadvantages of commonly used cost indicators and suggest reporting and plotting curves with all available cost indicators. With regard to calculating FLOPs, they criticize that these values are often theoretical values that do not take into account practical factors, such as the degree of parallelism. It provides a hardware-independent comparison, however, it does not translate to the speed of the model. Clark et al. [13] introduced ELECTRA, an approach to efficiently learn an encoder that classifies token replacements accurately. Therefore, they compare their approach to various transformer-based models on the GLUE benchmark. They do this by displaying the GLUE score along with FLOPs used in the pre-training process. Furthermore, they stress the importance of considering efficiency and call for future work on NLP to analyze computational use. Liu et al. [**liu_fastbert_2020**] calculate FLOPs to compare their model FastBERT to BERT, DistilBERT on sentence classification tasks and a sentences-matching task. Nagarajan et al. [40] report the average wall-clock time for their proposed

approach AxFormer and BERT-large.

3.2.2 Data

There exist several publicly available data sets that describe multi-label data (see 3.2).

Data set	#Classes	#Instances	Hierarchical?	Used by
Reuters-21578 (R21578)	90	10,788	N	[21, 22]
RCV1-V2	103	804,414	Y	[22]
EURLEX-4K [10]	3,993	15,539	N	[44]

Table 3.2: Public data sets

Commonly used data sets in the multi-class setting are BBC News (2,225 instances with 5 classes), and 20NewsGroup (18,846 instances with 20 classes) [6, 57].

3.2.3 Evaluation Frameworks

Galke et al. evaluated the multi-label data sets on the sample-based F1 measure, which is calculated separately for each example and then averaged. They argue that this metric addresses real-world applications where each document should be labeled as best as possible. To compare scores in existing research, they also consider Micro-F1 and Macro-F1. They manually search the best hyperparameters (taking 20% of the train set as validation set) resulting in a linearly decaying learning rate of 5e-5. Furthermore, they state that smaller batch sizes work better and therefore chose a batch size of 4 for 5 or 15 epochs. Galke et al., as well as Fallah et al. truncate their inputs to 512 tokens [21, 22]. If there is no further knowledge or tuning done for the hyperparameters of BERT and its variants, experiments often follow the recommendations made by Devlin et al. [18]: batch size=16,32, learning rate=5e-5, 4e-5, 3e-5, and 2e-5, and epochs=2,3,4.

In summary, the most frequent evaluation metrics for the performance in multi-label text classification are micro-F1 [21, 22, 46], sample-averaged F1 score [21, 22, 51], macro-F1 [46], Precision@k or Recall@k [44, 51, 63].

For multi-class text classification, Arslan et al. performed experiments for 1,3 and 5 fine-tuning epochs with the following fixed hyperparameters for fine-tuning: train and evaluation batch size of 16, maximum sequence length of 128, and adam learning rate of 4e-5. Wahba et al. [57] evaluated their

models on a train-test split ratio of 80:20 for a maximum of 3 epochs. There are slight modifications in the choice of hyperparameters (maximum sequence length of 256 and adam learning rate of 1e-5).

3.3 Summary

Since our research objectives deal with a sub-task of text classification, one cannot find exact research that covers the analysis of transformers with regard to performance and efficiency in (hierarchical) multi-label text classification. Most surveys only consider single-label text classification. Furthermore, comparison studies on multi-label classification using hierarchy-based methods are underrepresented [22]. Most similar objectives can be found in the research of Galke et al. [22]. However, they only state the number of model parameters as an indicator of the model's efficiency. As a result, we will follow research approaches that deal with the performance of transformers but include FLOPs to get additional insights into finding an optimal classification model.

Chapter 4

Methodology

In the following chapter, we will present our approach for comparing classification models in a HMTC setup. First, we will provide an understanding of the data by outlining the use case, listing the results of the exploratory data analysis, and describing the preprocessing of the data. Second, we will describe the models we choose and how we conduct the training including hyper parameter- and fine-tuning. Finally, we will present our evaluation framework to answer research question **RQ2**.

4.1 Data Understanding

The data set used in this thesis was provided by Ymner. Ymner is a Swedish company that connects entrepreneurs, researchers, and idea developers with public funders, investors, foundations, and funds to simplify innovation and research funding. It collects and analyses data on calls for proposals and grants from Sweden's public and private funders of research and innovation. Project proposals should follow a template that, among other guidelines, includes a research classification established by SCB (Statistics Sweden). An example of a funded research proposal can be found in Appendix A. Every project proposal in research should be classified with the corresponding area of research. The *standard for the Swedish classification of research subjects 2011*[1] is a classification scheme that was established by SCB and Högskoleverket (now Universitetskanslersämbetet) in 2010 and was last updated in 2016. It is used in official statistics to report doctoral students, higher education personnel, and research and development income by research subject. The SCB classification scheme is a hierarchical taxonomy consisting of three levels of classification.

[1] Documentation of SCB can be found here.

The highest level comprises six research subject areas: natural sciences [1], engineering [2], medicine and health sciences [3], agricultural and veterinary sciences [4], social sciences [5], and humanities and the arts [6]. Under each of these six research subject areas, there are five to eleven research subject groups, totaling 42 groups. The third level contains the research topic resulting in additional 260 labels. For each classification level, the research subjects are mapped to integer IDs that consist of 1, 3, or 5 digits, where IDs with 5 digits belong to the most specific level, i.e. the leaves of the classification tree. In summary, the SCB classification can be characterized as:

Level	IDs	#Labels
1	1-6	6
2	101-605	42
3	10101- 60599	260

Table 4.1: Characteristics of the SCB classification standard

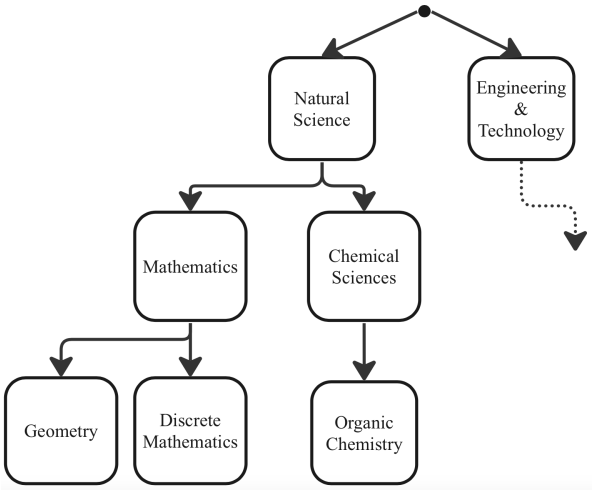


Figure 4.1: Example: Hierarchical classification structure that includes different levels of research fields.

Figure 4.1 illustrates part of the SCB structure. Suppose that a document belongs to the label *Geometry* [ID: 10102], we can conclude that it also belongs to *Mathematics* [ID: 101] and subsequently to *Natural Science* [ID: 1]. It is worth mentioning that each parent node has a child node that is denoted as *Other...* to cover the cases where the research does not belong to any other given opportunity. For example, one can find *Other Natural Science* [107]

under *Natural Science* [1] and the same yields for *Other Chemistry Topics* [10499] under *Chemical Sciences* [104].

The data consists of funded research projects. For the classification model, we make use of the following features of a research project: title, description, and corresponding SCB classifications. The project proposals are written in English and Swedish. However, we will focus only on English in the scope of the thesis to employ a single monolingual language model.

According to the official SCB standard, all objects to be classified should only belong to one research subject. The research topics included in the standard are thus "mutually exclusive". Subjects that are more interdisciplinary in nature should be classified under the research subject that constitutes the main part of the interdisciplinary subject. Although SCB demands this structure, the data provided by Ymner contains research projects that contain labels belonging to multiple research areas. In other words, a research project can be labeled with multiple labels (e.g. *Geometry* [10102] and *Zoology* [10608]) from the same level of classification. Figuratively, this corresponds to having multiple paths in the tree-based classification structure. In conclusion, the research classification task based on the Ymner data corresponds to a multi-path HMTC task.

4.1.1 EDA

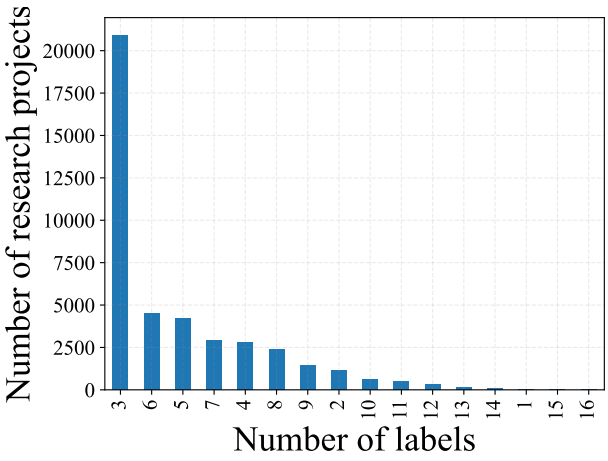
The original data set provided by Ymner consists of 59,220 research projects, of which 9,289 are unclassified and therefore directly removed. In addition, there are data points that do not contain any or enough information on the features that are relevant to us. First, we analyze the number of words in the features *description* and *title* of the research projects. One can find 792 research projects that only contain one word in the title. However, these titles often relate to an abbreviation of the project, such as "CAMART2". There also exist data points that do not contain any useful information about the description of the research project. For example, some descriptions say "See application", "Confidential application" or "To be developed". Counting the total number of words for both features leads to the statistic described in Table 4.2. To get sufficient information, we remove data points that contain less than 20 words in the title and description combined. We end up with a data set of 41,997 research projects.

After analyzing the text features, we now move our focus to the labels. Initially, we verify if each label within a research project correctly defines a path in the classification structure, meaning that the parenting label should

	Min	Max	μ	σ^2
#Words	4	3810	233.17	85.98

Table 4.2: Statistics for the total number of words from the title and description of the research project combined. The average is $\mu = 233.17$ and the variance $\sigma^2 = 85.98$.

also exist in the list of labels. To align with the path structure, we have to add 1,523 labels resulting in 203,114 labels in total. Having the final data set, we analyze the label distribution. Recall that we are dealing with a Hierarchical Multi-Label Text Classification (HMTC) task, which means that a research project can be classified with multiple labels from different classification levels. Figure 4.2 illustrates how many labels are assigned to a research project. If we disregard the classification level, a research project is assigned to on average $\mu = 4.66$ labels with a variance of $\sigma^2 = \pm 2.29$. More details can be found in Figure 4.2.



Level	μ	σ^2
1	1.30	0.57
2	1.58	0.87
3	1.92	1.17
-	4.66	2.29

Figure 4.2: Number of labels for one research project. The figure describes the overall number of labels indicating that most research projects are assigned 3 labels. The table summarises the mean and standard deviation with regard to the level of classification. For example, a research project is classified into an average of $\mu = 1.92$ classes of the most detailed level (5-digit level).

Figure 4.3 describes the year of funding. However, 16.01% of the research projects do not contain information about the year of funding.

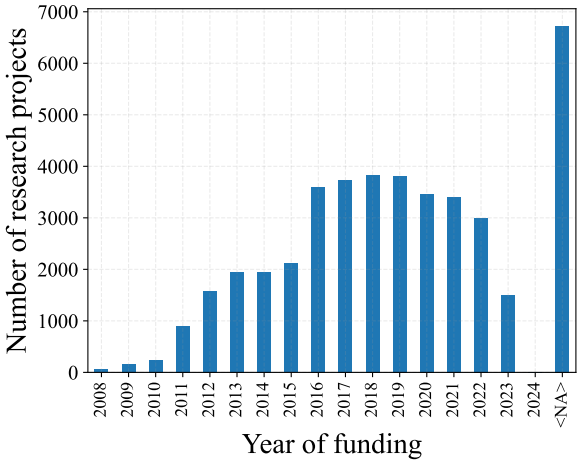


Figure 4.3: Distribution of funding years of the research projects.

When dealing with multi-label classifications, the distribution of labels often follows a long-tail distribution meaning that there exist labels that are used much more frequently than other labels. Looking at Figure 4.4 that shows a sorted count of the label IDs of classification level 1, we see that the label *Engineering and Technology [2]* was used the most. In contrast, *Agricultural and Veterinary Sciences* [4] and *Humanities and the Arts* [6] do not occur as often.

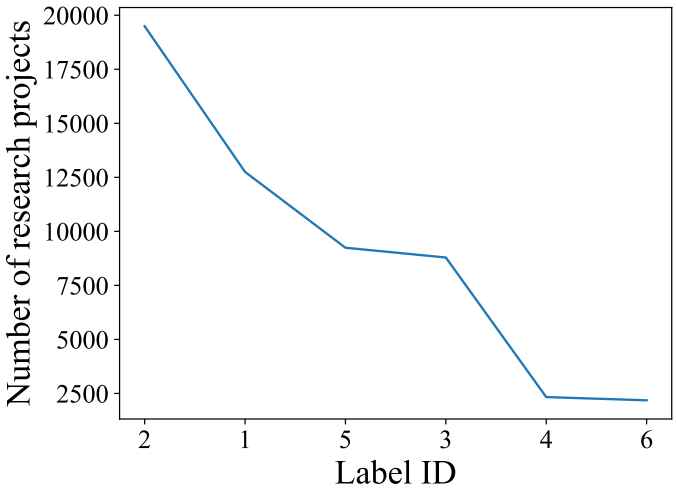


Figure 4.4: Distribution of labels for classification level 1.

Looking at the next classification levels in Figure 4.5, we see the same phenomena of a long tail distribution. The label *Pharmaceutical Chemistry*

[20404] was only used four times in the whole data set. Furthermore, there are several SCB classifications that could not be found in this data set: *Bioethics* [20804], *Psychology (excluding applied psychology)* [50101], *Law (excluding law and society)* [50501].

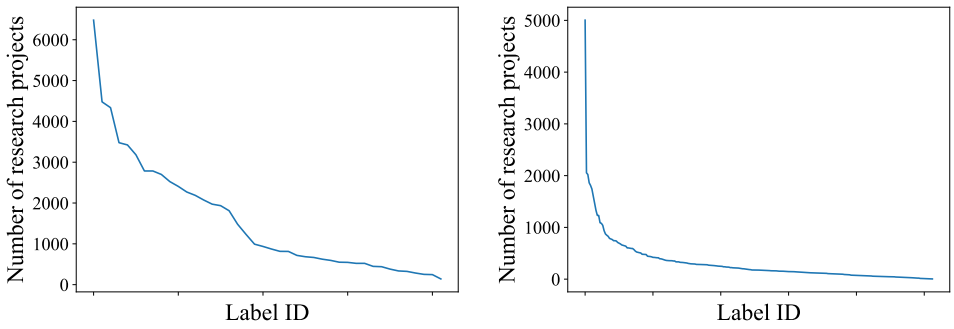


Figure 4.5: Long-tail distribution of labels of classification level 2 and level 3.

4.1.2 Preprocessing

The research projects are described by a title and a description, which we combine to form a data column containing the text data. Therefore, we join both strings by including a line break. To provide sufficient information for the text classifiers, we decide to filter out data points with less than 20 words, resulting in the removal of 79 data points. On top of that, we remove research projects that were labeled as *Unclassified*. We end up having 41,997 data points that cover 305 out of all 308 possible labels. For fine-tuning, we use 80% of the data which we divide into 64% for training and 16% for validation. The remaining 20% are used for the final evaluation and comparison. Table 4.3 describes the resulting data sets.

	Training	Validation	Testing
#Research Projects	26,877	6,720	8,400

Table 4.3: Sizes of data splits.

For the labels, we apply MultiLabelBinarizer() from scikit-learn to encode the target variables. Therefore, we fit the binarizer to the training data only and then transform the labels in all datasets.

Depending on the type of model, one needs to preprocess the textual data differently. In order to employ logistic regression as our baseline model, we additionally need to utilize feature extraction. The first step of preprocessing

is to define a regular expression tokenizer using the RegexpTokenizer function from the nltk.tokenize module. The regular expression used in this function specifies that only alphabetical characters and the "@" symbol are to be considered valid tokens. Additionally, we apply stemming and remove stop words to reduce noise in the data and improve the accuracy of the classifier. To reduce computational complexity and focus on words that contain useful information for the classification task, we restrict the number of features to the 10,000 most frequent tokens. Thus, the feature space has a dimension of 10,000 instead of 1,542,880, which would be the result if all tokens were retained. The choice of 10,000 is based on former research and intuition. Suneera et al. [52] applied logistic regression to a multiclass classification task for both the full feature space of 130,107 distinct words and a reduced version of the 5,000 most relevant words. Compared to a classification accuracy of 82.74 for the full feature space, the reduced dimension results in 78.98. Similarly, Madhfar et al. [33] studied the use of the most frequent n words as feature space for a data set consisting of 1,035,881 unique words in Arabic. They conclude that using more than 3,000 features has no positive effect on the performance of the models. Based on these findings and comparing their initial feature space to our space, we decide on 10,000 as a good starting point that ensures that enough important features are present while having a lower computational complexity than originally. After defining the feature space on the training data, we create the BOW representation of each research project to obtain numerical values.

In the case of the transformers, the raw data is tokenized using one of the tokenizer classes BertTokenizerFast and DistilBertTokenizerFast that are provided by Hugging Face.[1] They are based on WordPiece with a vocabulary size of 30,000 and split words into subwords allowing the model to handle out-of-vocabulary words. The tokenizers also perform a number of additional steps, such as lowercasing the text, adding special tokens like [CLS] and [SEP] to mark the beginning and end of a sequence, and mapping the tokens to integer IDs that can be understood by deep learning models. Hereby, we use a fast version of the original BertTokenizer or DistilBertTokenizer, which is more efficient due to an optimized implementation of the tokenization algorithm. The final preprocessing step is to store the data on disk, which allows faster data loading during training.

[1] https://huggingface.co

4.2 Modeling

This section provides an overview of the models selected for the comparison. In addition, the corresponding training processes including hyperparameter- and fine-tuning are outlined.

4.2.1 Baseline Model

We choose logistic regression as the baseline model because of its simplicity and ease of implementation. The simplicity of the baseline model allows for a clear understanding of the impact of more complex models on the classification task, making it an ideal starting point for evaluating models in our setting.

Although logistic regression is originally a binary classifier, we can convert the multi-label problem into a set of binary classification problems that can then be handled using logistic regression. For this purpose, we use the scikit-learn library to apply a classifier chain based on logistic regression as the base estimator. Since classifier chains are able to account for label dependencies, we can take advantage of the hierarchical taxonomy. By doing so, we pass the hierarchical structure—starting from the roots, i.e. the least specific research areas, up to the leaves—to the classification chain so that the classifiers can use the prediction from the higher-level nodes.

Although we primarily use a classifier chain, we can still fine-tune the hyperparameters used in logistic regression. We apply a grid search to the choice of penalty term and to the inverse of the regularization strength (smaller values mean stronger regularization). Based on published implementations, we choose the following possibilities: r(w) = ['l1', 'l2', 'elasticnet']; C = [0.1, 1, 10]. To find the best combination of hyperparameters, we follow a grid search approach and validate the classifiers' performance on the validation set.

4.2.2 Transformers

Our implementation for the transformers is based on the NLP library Simple Transformers[1] that is built on the work of Hugging Face and their Transformers library. We choose the model types BERT and DistilBERT. Both follow the same workflow of loading a checkpoint from a pre-trained model, fine-tuning the model on labeled data, and finally using the model to get predictions

[1] https://simpletransformers.ai

that are used for the evaluation. Due to computational constraints, we do not perform hyperparameter-tuning and instead rely on choices for the hyperparameters made in past research.

For the training process, we randomly sample and load the preprocessed training data into batches of batch size set to 4 (as in [22]). We use AdamW as an optimizer in the training process with a learning rate of 5e-5 for 15 training epochs. Furthermore, we adjust the learning rate throughout training by adding a linear rate schedule with a warm-up. This schedule starts with a low learning rate, gradually increases it to a maximum value, and then gradually decreases it again. The warmup period helps prevent the model from getting stuck in a poor local minimum during the early stages of training. We use *BCEWithLogitsLoss* as a loss function, which calculates the binary cross-entropy loss between the predicted and actual labels. During training, we evaluate the model's performance on a validation set. By calculating metrics such as LRAP and differently aggregated F1-scores, we can access the performance of the model and ensure that is not overfitting on the training data. On top of that, we apply early stopping with a waiting period of 3 epochs and a delta value of 0. This means the training process is stopped when the validation loss stops improving. The waiting period of 3 epochs ensures that the model has time to recover from small fluctuations in the validation loss, while the delta value of 0 determines the minimum amount of improvement required to continue training.

To get predictions, the model relies on the classification head on top of the regular transformer model block. It is a single-layer feed-forward network with an input size equal to the hidden size and an output size corresponding to the number of labels. The resulting logits of the main model block are applied to a sigmoid function to get the classification probabilities. Labels that have a classification probability larger than the threshold of 0.5 are assigned to the data point.

Table 4.4 lists the transformer models we utilized and summarises their model architecture. For BERT and DistilBERT, we select the uncased version which means that it does not distinguish between upper and lower case. Words such as "english" and "English" are treated the same. For BERT, we decide to test a maximal sequence length of 512 and additionally 128. We want to investigate the impact of sequence length on the model's performance. A shorter sequence length has a positive impact on the efficiency of the model, but it is not clear how much information is lost due to truncation. When using a maximum sequence length of 512 tokens, 990 research projects are truncated, compared to 39,660 truncated research projects when the maximum sequence

length is 128.

Model Type	Specification	#Layers	Hidden Size	#Attention Heads	#Parameters
BERT	bert-base-uncased	12	768	12	110 M
DistilBERT	distilbert-base-uncased	6	768	12	66 M

Table 4.4: Properties of the transformer models that we choose to investigate.

4.3 Evaluation Framework

To conduct a comparative study on the proposed classification models, we use a two-sided evaluation framework. On the one hand, we compare the performance of the models by evaluating the predictions on the test set. On the other hand, we calculate the FLOPs of running an inference of the model.

4.3.1 Performance

As discussed in Section 2.5, there exist several metrics to summarize the quality of classifications in a multi-label framework. The Simple Transformers library suggests the use of LRAP, which aims to give a better rank to the labels associated with each sample. Galke et al. [22] argue that "sample-based F1 represents real-world applications where each document needs to be annotated as well as possible". We opt to look primarily at sample-based F1, but we will also report LRAP and macro-F1 and micro-F1 following previous research. If the F1 score is poorly defined, meaning that some labels are not found in the predictions, the F1 score for that label is set to 0. each time specifying a different random seed for the data splits and the process of hyperparameter or fine-tuning of the model.

4.3.2 Usage Cost

For the evaluation of the usage cost of the classification models, we will use FLOPs to provide an objective metric without having to standardize computational and hardware resources. In the whole process of deploying classification

models and transformers, in particular, there are many computational parts for which one can calculate FLOPs. Since we use pre-trained language models, it is not feasible in the context of this work to compute FLOPs for training. Computing FLOPs for fine-tuning could also be an option, however, we focus on the inference cost. It contributes to the overall cost of a model especially with regard to long-term deployments. Additionally, this way we can compare it to the baseline, which does not rely on fine-tuning.

The classifier chain in our baseline model consists of multiple logistic regression classifiers that share their predictions iteratively. Thus, we first derive the FLOPs needed for a binary logistic regression model that predicts the classification probability of one class.

Strictly speaking, logistic regression requires a certain number of FLOPs for feature extraction. Preprocessing of the input data, including tokenization and subsequent transformation of the textual data into a BOW representation, is required before performing any inference. However, we were unable to find research investigating this part of the computation, nor useful information about the FLOPs used in feature extraction. As a result, we restrict our analysis to the assumption that we start with data that already consists of numerical features.

As derived in Equation (2.1), predicting the classification probability of a single class X_i is $p(X_i) = \sigma(X_i w + w_0)$ where the size of the feature and weight vector depends on the dimension of the feature space. For a data point of d features, a logistic regression model thus requires $2d + 1$ floating point operations to compute the dot product between the input features and the model weights including the bias term [64]. Additionally, the sigmoid function $f(z) = 1/(1 + exp(-z))$ is applied to the output that requires a constant number of operations. We estimate that by using 4 FLOPs: Multiplying d by -1, taking the exponential function, adding both terms and finally taking the reciprocal value. Thus, the total number of FLOPs for a binary prediction is:

$$\text{FLOPs}_{binary} = 2d + 5 \tag{4.1}$$

The classifier chain that we used for the baseline model consists of $k = 305$ binary logistic regression models. Each subsequent model takes the output of the previous model as an additional input feature. Therefore, the second model takes $(d + 1)$ features as an input and the third model $(d + 2)$, and so on, up to the k-th model that receives $(d + k - 1)$ features. The total number of floating

point operations required to predict a data point with the classifier chain is

$$\text{FLOPs}_{chain} = \sum_{i=0}^{k-1} 2(d+i) + 1 = k(k+2d) \tag{4.2}$$

To obtain FLOPs for the transformers, we can either rely on estimations found in previous research or compute them in the inference pipeline. We chose the latter option and use the code based on [13] that introduced the ELECTRA model and conducted an efficiency study. Appendix B lists the Python implementation that we use to calculate FLOPs. Clark et al. compose the total FLOPs for running an inference for a classification task as:

$$\text{FLOPs}_{total} = l \cdot \text{FLOPs}_{block} + \text{FLOPs}_{embedding} + \text{FLOPs}_{classification} \tag{4.3}$$

where l is denoted as the number of layers in a transformer.

- FLOPs_{block}: FLOPs required for the forward pass of a single transformer block. The calculations are based on the number of heads, the key/query/value dimension, and the intermediate dimension. The function calculates FLOPs for different parts of the transformer block, including the attention mechanism, the intermediate feedforward layer, and the output layer. Finally, the function returns the total FLOPs for the block multiplied by the sequence length of the input.
- $\text{FLOPs}_{embedding}$: FLOPs for the input embeddings. Specifically, FLOPs for the token-type embedding and position embedding, addition of these embeddings, layer normalization, and dropout are computed. Finally, the sum is multiplied by the sequence length of the input.
- $\text{FLOPs}_{classification}$: FLOPs for the linear classification layer.

Thereby, Clark et al. [13] make several assumptions. They define an operation as a mathematical operation, not a machine instruction, leading to the exponential function being treated as an addition, for example. They argue that matrix multiplication dominates the computing for most models anyway. They define several constants that describe FLOPs per neuron that are needed for frequently used computations. Computing dropout requires 4 FLOPs, activation 8 FLOPs (assuming GELU), layer norm, and softmax 5 FLOPs each.

However, Clark et al. [13] only compute the inference for binary classification. For our non-binary multi-label classification problem, we modify the

implementation such that $FLOPs_{classification}$ considers an output with a specified number of labels. We also need to add FLOPs for the application of the sigmoid function $f(x) = 1/(1 + exp(-x))$, which we estimated earlier with 4 FLOPs.

To be more precise, one would need to threshold the classification probabilities in both logistic regression and transformers to get the final predictions in terms of labels. However, since this is part of any classification model, we will discard these FLOPs. This decision is consistent with the approach of Clark et al. [13].

In addition to modifying the work of Clark et al., we adopt a commonly used estimate proposed in [25]. It estimates the FLOPs needed for a forward pass by considering the number of model parameters. We follow an implementation that can be found in the paper by Dürlich et al. [19].

Chapter 5

Results and Discussion

In this chapter, we present the results of our evaluation framework and benchmark performance against usage cost to find an optimal classification model for our use case.

5.1 Performance

To evaluate the performance of each classification model, we primarily compare F1-scores averaged on various subsets and additionally LRAP to provide multiple assessments on the quality of the classification task. For the experiments, we utilize the CPU for the baseline model and Nvidia Tesla V100 SXM2 32GB for the transformers. As mentioned earlier, we ran each experiment five times, changing the random seed for the data split and the model. For the baseline model, tuning the hyperparameters on the validation set resulted in choosing L2 regularization for three out of five runs and L1 regularization for the remaining two runs. The strongest regularization parameter $C = 0.1$ consistently yielded the best performance across all five runs. For the transformers, we only conducted fine-tuning with early stopping of the evaluation loss which is defined as cross-entropy. Figure 5.1 illustrates the evaluation loss for the three transformer models. BERT [512] (BERT with a maximal sequence length of 512) took the most steps to reach the early stopping condition.

Figure 5.2 illustrates the sample-averaged F1-score on the test set including all classification levels. We see that the baseline model and BERT [128] obtain a similar F1-score of 0.51. The best-performing model is DistilBERT with a score of 0.56. In order to see how well the models solve the classification task for each classification level, we calculate the F1-score for the corresponding

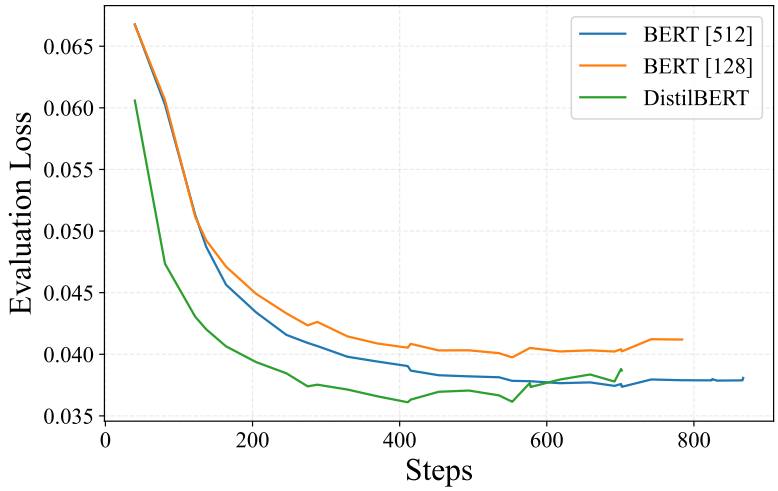


Figure 5.1: The evaluation loss (cross-entropy) during fine-tuning is averaged for all the runs. BERT is tested both for a sequence length of 128 and 512.

label subsets. The results are illustrated in Figure 5.3. Intuitively, assigning labels from the broadest classification level is the easiest task leading to performances above 0.74. Looking only at the transformers, we see that DistilBERT achieves the best F1-scores across all classification levels. However, the baseline model leads to the best F1-score of 0.33 for the most specific classification level. In comparison, the transformers reach from 0.22 to 0.31 where DistilBERT scores the best. In Appendix C, one can find the F1-scores for each classification layer and run.

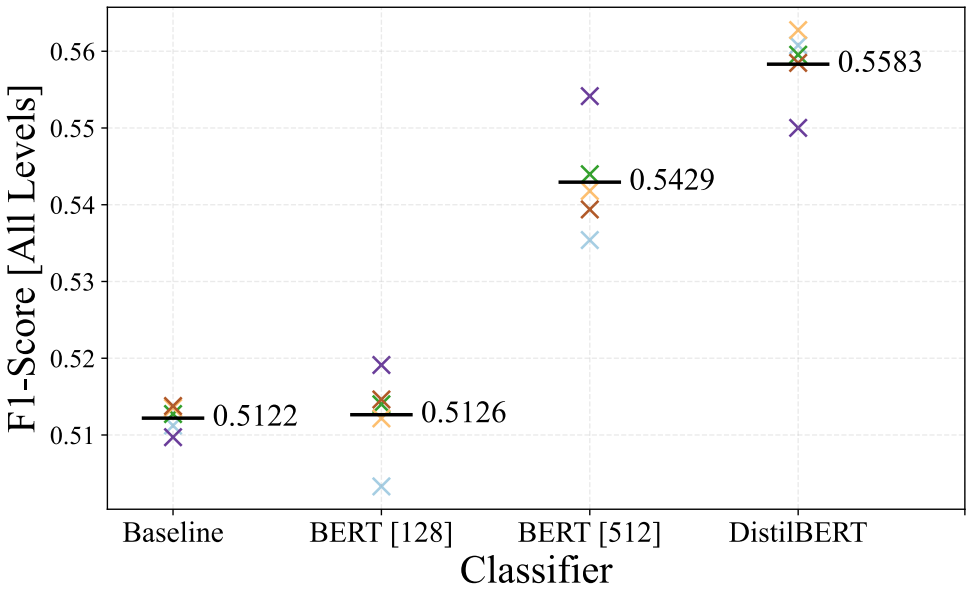


Figure 5.2: Sample-averaged F1-score on the test sets for each classifier, where each $\times$ displays one out of five random seeds. The mean is illustrated as the horizontal black line.

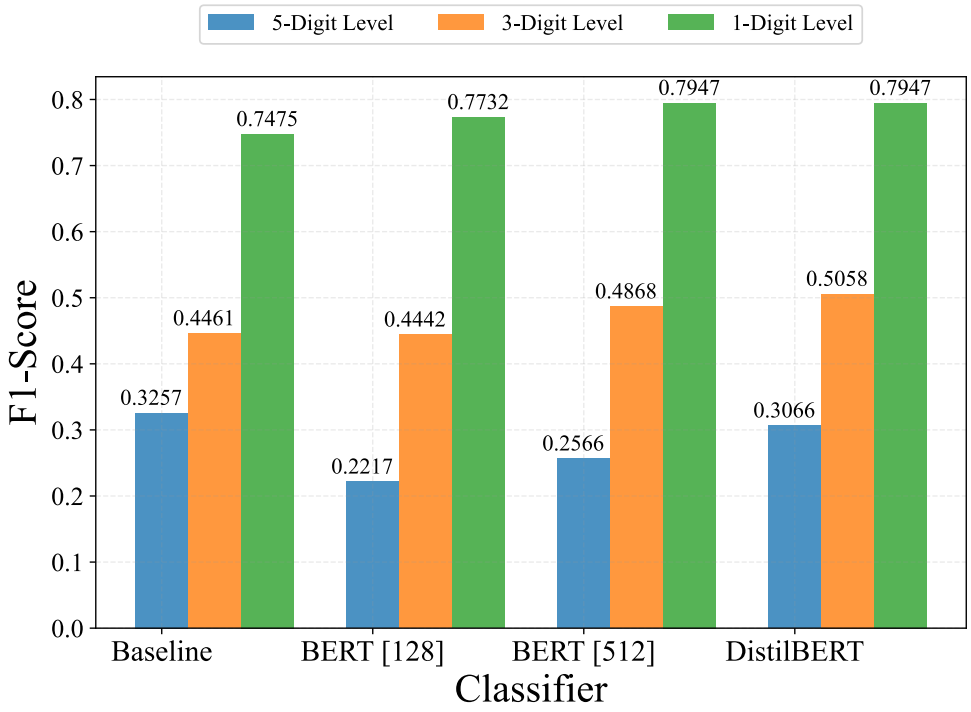


Figure 5.3: F1-scores averaged only on a selection of labels. It shows the three different classification levels represented by the number of digits used for the label IDs.

In addition to analyzing F1-score, we calculate the LRAP for the test set and the results are illustrated in Figure 5.4. The LRAP values achieved by the transformers range from 0.64 to 0.68, indicating a higher precision in ranking the labels correctly. On the other hand, the baseline model achieves a lower LRAP of 0.4, suggesting a lower precision in label ranking. This implies that the transformers, particularly the BERT models, have a better ability to accurately rank the labels in the test set compared to the baseline model.

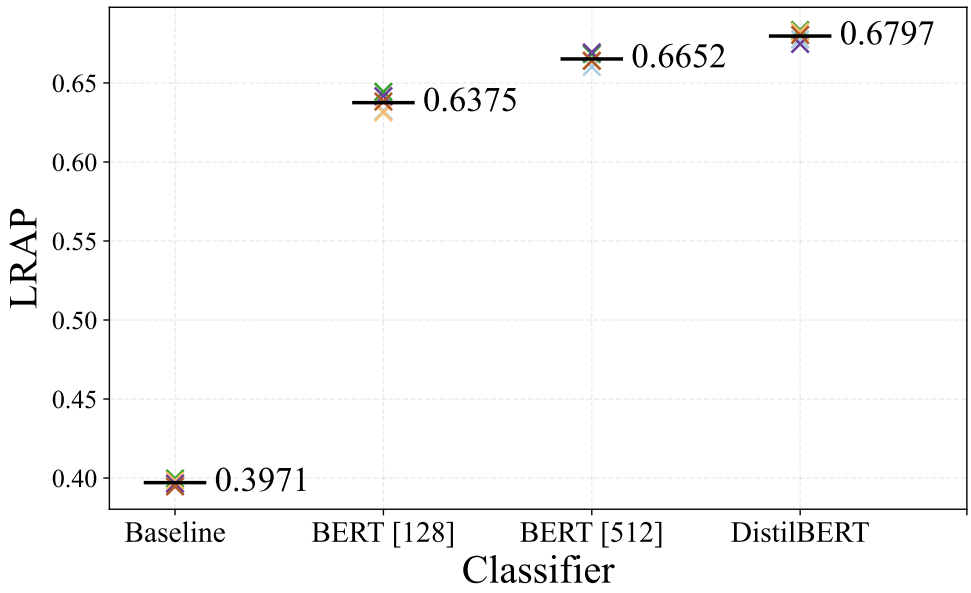


Figure 5.4: LRAP for each classifier for five random seeds.

5.2 Usage Cost

For the baseline model, we use the formula derived in Section 4.3.2 that does not include feature extraction. Since we set the vocabulary size to 30,000 for the vectorizer, we obtain a feature space of dimension $d = 10,000$. Furthermore, the data set consists of 305 labels leading to a classifier chain of $k = 305$ logistic regression models.

In addition to modifying the calculation from [13], we use the estimate of Kaplan et al. [25] and get the following estimations:

	FLOPs [1e6]			
	Our Estimation		Estimation from [25]	
Model		incl. Embedding		incl. Embedding
Baseline	**6**	-	-	-
BERT [128]	**22,619**	28,646	21,986	28,087
BERT [512]	**98,007**	122,419	87,942	112,350
DistilBERT	**49,367**	73,779	44,394	68,800

Table 5.1: FLOPs used for a single inference. The baseline model consists of a classifier chain based on logistic regression. The transformers use checkpoints of the pre-trained models bert-base-uncased and distilbert-base-uncased.

A comparison between our estimation and the estimation from Kaplan [25] reveals a significant degree of similarity. In our analysis, we focus on the estimates that exclude the operations of embedding, which allows for a fair comparison with the feature extraction of the baseline model. For the baseline model, a single inference only requires 6 million FLOPs, indicating its computational efficiency. Moving on to the BERT model with a maximum sequence length of 128 tokens, our estimation shows a higher FLOPs count of 22,619 million, reflecting the increased computational complexity. Similarly, the larger BERT model with a maximum sequence length of 512 tokens exhibits a significantly higher FLOPs count of 98,007 million. On the other hand, theDistilBERT model, which is characterized by its smaller and more efficient architecture compared to BERT, demonstrates a FLOPs requirement of 49,367 million according to our estimation.

5.3 Discussion

The two-sided evaluation framework allows for a comprehensive comparison of the classifiers. Figure 5.5 illustrates the side-by-side comparison of both metrics aggregated over all classification layers. Figures that illustrate the performance with regard to the 5-digit and 3-digit classification levels can be found under Appendix D. It is important to note that we utilized the same fine-tuning setup and transformer hyperparameters for consistency. Conducting hyperparameter tuning could potentially improve the performance of each transformer. Moreover, accurately determining the FLOPs of the baseline model poses challenges. As mentioned earlier, we did not consider the FLOPs required for feature extraction. While this aspect constitutes a significant portion of the computational cost for the baseline model, our estimation suggests that the total number of FLOPs for the baseline model is still expected to be significantly lower than that of the transformer.

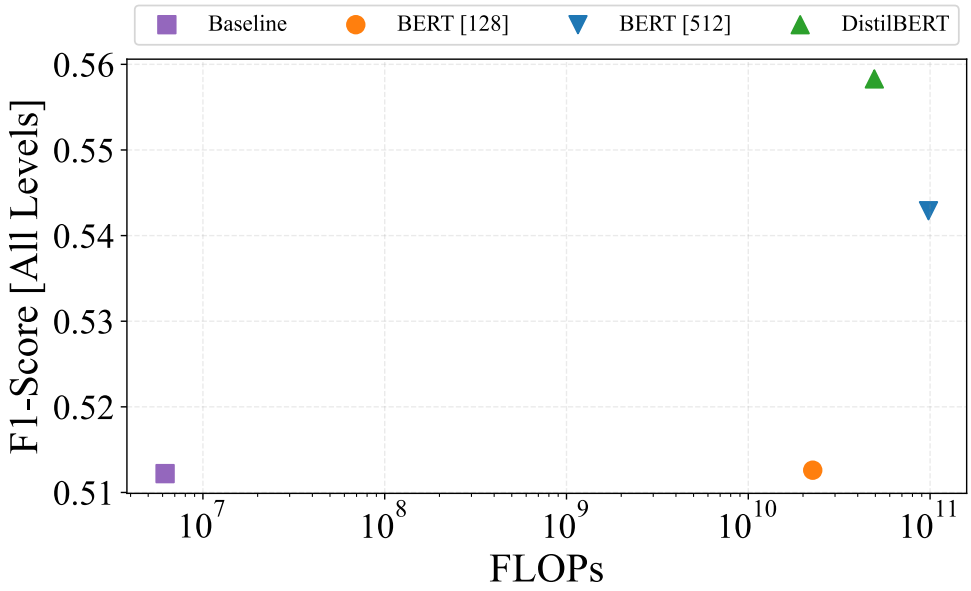


Figure 5.5: Comparison of performance and usage cost for each classifier. Here, the FLOPs are used without including the embedding and are scaled logarithmically.

The baseline model stands out for its significantly lower FLOPs usage; however, this comes at the expense of performance. Upon analyzing the transformers, we find that DistilBERT performs optimally within our fine-tuning setup. Despite requiring nearly double the FLOPs compared to BERT128, DistilBERT achieves approximately 0,05 performance improvement. Moreover, it turns out that BERT [512] cannot match the performance of DistilBERT. However, the base model of BERT consists of more parameters and could theoretically yield better results given the appropriate hyperparameters. Notably, the choice of maximum sequence length in BERT affects both usage cost and performance. Restricting the sequence to 128 tokens results in a sacrifice of more than 0,03 in performance but offers significant gains in usage cost. This finding indicates that the title and early sections of the research proposals lack sufficient information for the classification task.

By examining the different levels of hierarchical classification (refer to figures in Appendix C), we gain valuable insights into the specific capabilities of a pre-trained language model and a classical ML approach based on feature extraction. When classifying research proposals at the 1-digit level all the transformers, particularly DistilBERT and BERT512, exhibit an approximately 0,05 higher F1-score compared to the baseline model.

At the 1-digit level, the classification involves choosing from six labels that are commonly used in everyday language. The pre-trained language models, drawing on their broad knowledge from pre-training, can easily distinguish between the less overlapping categories. Similar to a person who does not need to have expertise in every field of research, they can nevertheless distinguish between research proposals related to *Natural Sciences* or *Social Sciences*.

However, as we delve into the most challenging classification level with 260 labels, we observe that the ML model performs better. It achieves an F1-score nearly 0.02 higher than DistilBERT and even 0.1 higher than BERT128. Despite fine-tuning, the language models struggle to differentiate between more detailed classifications. If we take the same example, it would be difficult for a person to distinguish between the terms *Discrete Mathematics* and *Computational Mathematics*. We believe that feature extraction allows the baseline model to uniquely associate terms and words that define the subcategories, while the pre-trained language models seem to struggle to leverage these subtle distinctions between research areas. Our reasoning aligns with related work in the domain of extreme multi-label classification, where ML-based models are considered to be a superior approach. Furthermore, the transformers we selected do not explicitly consider the hierarchical structure of the classifications. However, the classifier chain used for the baseline model is able to exploit the dependencies between labels.

In conclusion, our evaluation framework and benchmarking analysis have provided valuable insights into the performance and usage cost of various classifiers. We compared F1-scores and LRAP to assess the quality of the classification task. The transformers, particularly DistilBERT, achieved higher F1-scores and LRAP values compared to the baseline model, indicating their better ability to rank labels accurately. However, the baseline model demonstrated lower FLOPs usage and performed better in the most specific classification level. By examining hierarchical classification levels, we observed that pre-trained language models excel at broader classifications but struggle with more detailed ones.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis aimed to comprehensively compare discriminative language models for text classification. In the initial sections, we laid the foundation for classifying research proposals within a hierarchical classification structure. We discussed both a classical ML-based approach and our adaptation of the classification setup using transformers. By formulating the theoretical approach for addressing a hierarchical text classification task and subsequently executing the classification task, we answered research question **RQ1**. Next, we introduced our evaluation framework, focusing on quantifying the usage cost by calculating the FLOPs required for inference. Thus, we have effectively tackled research question **RQ2**. Upon conducting experiments, we presented the results obtained from our evaluation framework. The evaluation delved deeper into the comparison of usage cost and performance, with DistilBERT exhibiting the best balance within the defined fine-tuning setup for the overall performance aggregated for all classification levels. This serves as our response to research question **RQ3**. Furthermore, we gained valuable insights into the specific capabilities of pre-trained language models and a classical ML-based approach, depending on the complexity of the classification task.

One aspect of the thesis focused on fine-tuning the transformers and conducting hyperparameter tuning, as well as training the logistic regression model to obtain performance metrics. Another aspect involved analyzing the usage cost, for which we utilized existing research and implementations to calculate the FLOPs required for inferences of a transformer model. However, we proposed a modified FLOPs estimate to account for our specific classification problem, allowing for the inclusion of a multi-label classification layer in the

estimate.

In summary, this research contributes to the understanding of discriminative language models for text classification, providing valuable insights into their performance, usage cost, and applicability in different classification scenarios. The findings presented here can inform future research and guide the selection of suitable models and approaches for similar tasks in the field of NLP.

6.2 Limitations and Future Work

In the following section, we will discuss the limitations inherent in this thesis, which provide valuable insights into the boundaries and constraints that influenced our research findings and interpretations. By acknowledging and addressing these limitations, we aim to provide a comprehensive understanding of the scope and validity of our work. Furthermore, we will outline potential directions for future research, exploring areas that extend beyond the limitations of this study.

In our study, we decided to utilize the BERT base model with two different maximal sequence lengths and DistilBERT as our discriminative language models. However, to conduct a more thorough analysis of the trade-off between usage cost and performance, it would have been beneficial to consider additional transformer models, particularly those optimized to be more efficient. Furthermore, an intriguing possibility for future work would involve evaluating the performance of SciBERT [7], a variant of BERT specifically pre-trained on scientific language. Given that our study focused on research proposals that featured scientific terms, utilizing a language model with a corresponding vocabulary could potentially yield enhanced performance. In addition to the choice of transformers, we acknowledge that conducting hyperparameter tuning for each model could have further optimized the fine-tuning setup. By individually fine-tuning each model and carefully selecting the optimal hyperparameters, we could have potentially improved the performance of the models and obtained more refined results.

When considering our evaluation framework, it is worth noting that we could have incorporated a broader range of metrics to gain additional insights into both the performance and usage cost aspects. Furthermore, to enhance the robustness of our findings, we could have conducted more than five runs with different random seeds for both the data split and model initialization. This would have allowed us to observe the potential variations in the results and assess the stability and consistency of our findings. Alternatively, we could

have performed multiple experiments while maintaining the same data split and altering the model initialization to observe the impact on the results.

In terms of the FLOPs calculation for the logistic regression model, it is important to acknowledge that our analysis did not take into account the FLOPs associated with transforming the textual data into features that can be processed by the ML model. It is worth noting that FLOPs calculations are primarily used in the context of DL models, and thus, we encountered a lack of existing research specifically addressing the FLOPs for feature extraction in traditional ML models. We believe that this limitation stems from the fact that the computations for feature extraction can vary significantly depending on the specific ML library and its implementation.

Lastly, we used one dataset that contains a hierarchical classification structure. While this dataset served as a valuable basis for our comparative study, it is important to acknowledge the potential limitations associated with using only one dataset. Additionally, we chose transformer models that do not directly exploit the hierarchical information and label dependencies present in the dataset. It would have been intriguing to explore the impact of incorporating the hierarchical structure into the fine-tuning process of the transformer models. As highlighted in Section 3.1, there exist transformers that are capable of incorporating hierarchical information. This opens up an interesting avenue for future research. Another possibility for the analysis within the hierarchical classification structure is the re-weighting of classification levels during the fine-tuning process, as proposed by Lu et al. [32]. Moreover, one could consider re-weighting the aggregated F1 score for the final performance evaluation.

References

[1] *1.1.11. Logistic regression*. en. URL: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (visited on 04/08/2023).

[2] *1.12. Multiclass and multioutput algorithms — scikit-learn 1.2.2 documentation*. URL: https://scikit-learn.org/stable/modules/multiclass.html (visited on 04/08/2023).

[3] *3.3. Metrics and scoring: quantifying the quality of predictions*. en. URL: https://scikit-learn/stable/modules/model_evaluation.html (visited on 04/08/2023).

[4] Charu C. Aggarwal and ChengXiang Zhai. "A Survey of Text Classification Algorithms." en. In: *Mining Text Data*. Ed. by Charu C. Aggarwal and ChengXiang Zhai. Boston, MA: Springer US, 2012, pp. 163–222. ISBN: 978-1-4614-3222-7 978-1-4614-3223-4. DOI: 10.1007/978-1-4614-3223-4_6. URL: http://link.springer.com/10.1007/978-1-4614-3223-4_6 (visited on 01/31/2023).

[5] *Amazon EC2 Update – Inf1 Instances with AWS Inferentia Chips for High Performance Cost-Effective Inferencing | AWS News Blog*. en-US. Section: Amazon EC2. Dec. 2019. URL: https://aws.amazon.com/blogs/aws/amazon-ec2-update-inf1-instances-with-aws-inferentia-chips-for-high-performance-cost-effective-inferencing/ (visited on 04/19/2023).

[6] Yusuf Arslan, Kevin Allix, Lisa Veiber, Cedric Lothritz, Tegawendé F. Bissyandé, Jacques Klein, and Anne Goujon. "A Comparison of Pre-Trained Language Models for Multi-Class Text Classification in the Financial Domain." en. In: *Companion Proceedings of the Web Conference 2021*. Ljubljana Slovenia: ACM, Apr. 2021, pp. 260–268.

ISBN: 978-1-4503-8313-4. DOI: 10.1145/3442442.3451375. URL: https://dl.acm.org/doi/10.1145/3442442.3451375 (visited on 02/21/2023).

[7] Iz Beltagy, Kyle Lo, and Arman Cohan. *SCIBERT: A Pretrained Language Model for Scientific Text*. Tech. rep., pp. 3615–3620. URL: https://github.com/google-research/.

[8] Christopher M. Bishop. *Pattern recognition and machine learning*. en. Information science and statistics. New York: Springer, 2006. ISBN: 978-0-387-31073-2.

[9] Tom B Brown et al. "Language Models are Few-Shot Learners." In: (2020). URL: https://arxiv.org/abs/2005.14165.

[10] Ilias Chalkidis, Emmanouil Fergadiotis, Prodromos Malakasiotis, and Ion Androutsopoulos. "Large-Scale Multi-Label Text Classification on EU Legislation." In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 6314–6322. DOI: 10.18653/v1/P19-1636. URL: https://aclanthology.org/P19-1636 (visited on 04/13/2023).

[11] Haibin Chen, Qianli Ma, Zhenxi Lin, and Jiangyue Yan. "Hierarchy-aware Label Semantics Matching Network for Hierarchical Text Classification." In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 4370–4379. DOI: 10.18653/v1/2021.acl-long.337. URL: https://aclanthology.org/2021.acl-long.337 (visited on 03/23/2023).

[12] François Chollet. *Deep learning with Python*. en. OCLC: ocn982650571. Shelter Island, New York: Manning Publications Co, 2018. ISBN: 978-1-61729-443-3.

[13] Kevin Clark, Minh-Thang Luong, and Quoc V Le. "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators." en. In: (Mar. 2020). DOI: 10.48550/arXiv.2003.10555. URL: http://arxiv.org/abs/2003.10555.

[14] Eduardo Costa, Ana Lorena, Andre de Carvalho, Alex Freitas, and Nicholas Holden. “Comparing Several Approaches for Hierarchical Classification of Proteins with Decision Trees.” In: Jan. 2007, pp. 126–137.

[15] *Deep Learning (Adaptive Computation and Machine Learning series): Goodfellow, Ian, Bengio, Yoshua, Courville, Aaron: 9780262035613: Amazon.com: Books*. URL: https://www.amazon.com/Deep-Learning-Adaptive-Computation-Machine/dp/0262035618/ref=sr_1_1?ie=UTF8&qid=1472485235&sr=8-1&keywords=deep+learning+book (visited on 03/07/2023).

[16] Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. *The Efficiency Misnomer*. arXiv:2110.12894 [cs, stat]. Mar. 2022. URL: http://arxiv.org/abs/2110.12894 (visited on 02/02/2023).

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “ImageNet: A large-scale hierarchical image database.” In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. ISSN: 1063–6919. June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[18] Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” In: *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference* 1 (Oct. 2018). Publisher: Association for Computational Linguistics (ACL), pp. 4171–4186. ISSN: 9781950737130. DOI: 10.48550/arxiv.1810.04805. URL: https://arxiv.org/abs/1810.04805v2.

[19] Luise Dürlich, Evangelia Gogoulou, and Joakim Nivre. “On the Concept of Resource-Efficiency in NLP.” In: *Proceedings of the 24th Nordic Conference on Computational Linguistics (NoDaLiDa)*. Tórshavn, Faroe Islands: University of Tartu Library, May 2023, pp. 135–145. URL: https://aclanthology.org/2023.nodalida-1.15 (visited on 06/13/2023).

[20] Matúš Falis, Hang Dong, Alexandra Birch, and Beatrice Alex. *CoPHE: A Count-Preserving Hierarchical Evaluation Metric in Large-Scale Multi-Label Text Classification*. arXiv:2109.04853 [cs] version: 1. Sept. 2021. URL: http://arxiv.org/abs/2109.04853 (visited on 04/12/2023).

[21] Haytame Fallah, Patrice Bellot, Emmanuel Bruno, and Elisabeth Murisasco. "Adapting Transformers for Multi-Label Text Classification." en. In: (July 2022). URL: https://ceur-ws.org/Vol-3178/CIRCLE_2022_paper_07.pdf.

[22] Lukas Galke, Andor Diera, Bao Xin Lin, Bhakti Khera, Tim Meuser, Tushar Singhal, Fabian Karl, and Ansgar Scherp. *Are We Really Making Much Progress? Bag-of-Words vs. Sequence vs. Graph vs. Hierarchy for Single- and Multi-Label Text Classification*. arXiv:2204.03954 [cs] version: 2. Mar. 2023. URL: http://arxiv.org/abs/2204.03954 (visited on 04/12/2023).

[23] Palash Goyal, Sumit Pandey, and Karan Jain. *Deep Learning for Natural Language Processing*. en. Berkeley, CA: Apress, 2018. ISBN: 978-1-4842-3684-0 978-1-4842-3685-7. DOI: 10.1007/978-1-4842-3685-7. URL: http://link.springer.com/10.1007/978-1-4842-3685-7 (visited on 01/30/2023).

[24] Ting Jiang, Deqing Wang, Leilei Sun, Zhongzhi Chen, Fuzhen Zhuang, and Qinghong Yang. *Exploiting Global and Local Hierarchies for Hierarchical Text Classification*. arXiv:2205.02613 [cs]. Nov. 2022. DOI: 10.48550/arXiv.2205.02613. URL: http://arxiv.org/abs/2205.02613 (visited on 04/13/2023).

[25] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. *Scaling Laws for Neural Language Models*. arXiv:2001.08361 [cs, stat]. Jan. 2020. URL: http://arxiv.org/abs/2001.08361 (visited on 02/01/2023).

[26] Kamran Kowsari, Donald E. Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S. Gerber, and Laura E. Barnes. "HDLTex: Hierarchical Deep Learning for Text Classification." In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. Dec. 2017, pp. 364–371. DOI: 10.1109/ICMLA.2017.0-134.

[27] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura E. Barnes, and Donald E. Brown. "Text Classification Algorithms: A Survey." In: *Information* 10.4 (Apr. 2019). arXiv:1904.08067 [cs, stat], p. 150. ISSN: 2078-2489. DOI: 10.3390/info10040150. URL: http://arxiv.org/abs/1904.08067 (visited on 01/31/2023).

[28] CNPIEC KEXIN LTD. *Datasets for NLPCC2022.SharedTask5.Track1*. en. 2022. DOI: 10.11922/sciencedb.j00104.00100. URL: https://www.scidb.cn/en/detail?dataSetId=f34c8d1302204b50a293da1d2a11f701 (visited on 04/13/2023).

[29] George Leopold. *AWS to Offer Nvidia's T4 GPUs for AI Inferencing*. en-US. Mar. 2019. URL: https://www.hpcwire.com/2019/03/19/aws-upgrades-its-gpu-backed-ai-inference-platform/ (visited on 04/19/2023).

[30] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. "RCV1: A New Benchmark Collection for Text Categorization Research." en. In: *Journal of Machine Learning Research* 5 (Apr. 2004), pp. 361–397.

[31] Xin Liang, Dawei Cheng, Fangzhou Yang, Yifeng Luo, Weining Qian, and Aoying Zhou. "F-HMTC: Detecting Financial Events for Investment Decisions Based on Neural Hierarchical Multi-Label Text Classification." en. In: vol. 5. ISSN: 1045-0823. July 2020, pp. 4490–4496. DOI: 10.24963/ijcai.2020/619. URL: https://www.ijcai.org/proceedings/2020/619 (visited on 04/13/2023).

[32] Junyu Lu, Hao Zhang, Zhexu Shen, Kaiyuan Shi, Liang Yang, Bo Xu, Shaowu Zhang, and Hongfei Lin. "Multi-task Hierarchical Cross-Attention Network for Multi-label Text Classification." en. In: *Natural Language Processing and Chinese Computing*. Ed. by Wei Lu, Shujian Huang, Yu Hong, and Xiabing Zhou. Vol. 13552. Series Title: Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2022, pp. 156–167. ISBN: 978-3-031-17188-8 978-3-031-17189-5. DOI: 10.1007/978-3-031-17189-5_13. URL: https://link.springer.com/10.1007/978-3-031-17189-5_13 (visited on 03/23/2023).

[33] Mokhtar Ali Hasan Madhfar and Mohammed Abdullah Hassan Al-Hagery. "Arabic Text Classification: A Comparative Approach Using a Big Dataset." In: *2019 International Conference on Computer and*

Information Sciences (ICCIS). Apr. 2019, pp. 1–5. DOI: 10.1109/ICCISci.2019.8716479.

[34] Aditya Malte and Pratik Ratadiya. *Evolution of transfer learning in natural language processing*. arXiv:1910.07370 [cs]. Oct. 2019. URL: http://arxiv.org/abs/1910.07370 (visited on 03/13/2023).

[35] Christopher Manning, Prabhakar Raghavan, and Hinrich Schuetze. *Introduction to Information Retrieval*. en. Cambridge University Press, 2009. ISBN: 978-0-521-86571-5.

[36] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. arXiv:1301.3781 [cs]. Sept. 2013. URL: http://arxiv.org/abs/1301.3781 (visited on 08/22/2023).

[37] Shervin Minaee. "Deep Learning-based Text Classification: A Comprehensive Review." In: *ACM Comput. Surv* 54 (2021). DOI: 10.1145/3439726. URL: https://doi.org/10.1145/3439726.

[38] Thomas P Minka. "A comparison of numerical optimizers for logistic regression." en. In: *CMU Technical Report* (2003). URL: https://api.semanticscholar.org/CorpusID:15338459.

[39] Roberto Carlos Morales-Hernández, Joaquín Gutiérrez Jagüey, and David Becerra-Alonso. "A Comparison of Multi-Label Text Classification Models in Research Articles Labeled With Sustainable Development Goals." In: *IEEE Access* 10 (2022). Conference Name: IEEE Access, pp. 123534–123548. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3223094.

[40] Amrit Nagarajan, Sanchari Sen, Jacob R. Stevens, and Anand Raghunathan. "AxFormer: Accuracy-driven Approximation of Transformers for Faster, Smaller and more Accurate NLP Models." In: *Proceedings of the International Joint Conference on Neural Networks* 2022-July (2022). Publisher: Institute of Electrical and Electronics Engineers Inc. ISSN: 9781728186719. DOI: 10.1109/IJCNN55064.2022.9892797.

[41] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. *Carbon Emissions and Large Neural Network Training*. arXiv:2104.10350 [cs]. Apr. 2021. URL: http://arxiv.org/abs/2104.10350 (visited on 02/02/2023).

[42] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global Vectors for Word Representation." en. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: http://aclweb.org/anthology/D14-1162 (visited on 04/19/2023).

[43] Rafael B. Pereira, Alexandre Plastino, Bianca Zadrozny, and Luiz H. C. Merschmann. "Categorizing feature selection methods for multi-label classification." en. In: *Artificial Intelligence Review* 49.1 (Jan. 2018), pp. 57–78. ISSN: 1573-7462. DOI: 10.1007/s10462-016-9516-4. URL: https://doi.org/10.1007/s10462-016-9516-4 (visited on 04/23/2023).

[44] Mohammadreza Qaraei, Sujay Khandagale, and Rohit Babbar. "Why state-of-the-art deep learning barely works as good as a linear classifier in extreme multi-label text classification." en. In: *Computational Intelligence* (2020).

[45] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. *Learning to Generate Reviews and Discovering Sentiment*. arXiv:1704.01444 [cs]. Apr. 2017. URL: http://arxiv.org/abs/1704.01444 (visited on 03/14/2023).

[46] Rahul Ragesh, Sundararajan Sellamanickam, Arun Iyer, Ram Bairi, and Vijay Lingam. *HeteGCN: Heterogeneous Graph Convolutional Networks for Text Classification*. arXiv:2008.12842 [cs, stat]. Aug. 2020. URL: http://arxiv.org/abs/2008.12842 (visited on 04/21/2023).

[47] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. en. July 2020. DOI: 10.48550/arXiv.1910.10683. URL: http://arxiv.org/abs/1910.10683.

[48] Evan Sandhaus. *The New York Times Annotated Corpus*. Artwork Size: 3250585 KB Pages: 3250585 KB. Oct. 2008. DOI: 10.35111/77BA-9X74. URL: https://catalog.ldc.upenn.edu/LDC2008T19 (visited on 04/13/2023).

[49] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. arXiv:1910.01108 [cs]. Feb. 2020. DOI: 10.48550/arXiv.1910.01108. URL: http://arxiv.org/abs/1910.01108 (visited on 06/20/2023).

[50] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. "Green AI." In: *Communications of the ACM* 63.12 (Nov. 2020), pp. 54–63. ISSN: 0001-0782. DOI: 10.1145/3381831. URL: https://dl.acm.org/doi/10.1145/3381831 (visited on 04/19/2023).

[51] Zein Shaheen, Gerhard Wohlgenannt, and Erwin Filtz. *Large Scale Legal Text Classification Using Transformer Models*. arXiv:2010.12871 [cs]. Oct. 2020. URL: http://arxiv.org/abs/2010.12871 (visited on 02/20/2023).

[52] C M Suneera and Jay Prakash. "Performance Analysis of Machine Learning and Deep Learning Models for Text Classification." In: *2020 IEEE 17th India Council International Conference (INDICON)*. ISSN: 2325-9418. Dec. 2020, pp. 1–6. DOI: 10.1109/INDICON49873.2020.9342208.

[53] Yukihiro Tagami. "AnnexML: Approximate Nearest Neighbor Search for Extreme Multi-label Classification." en. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Halifax NS Canada: ACM, Aug. 2017, pp. 455–464. ISBN: 978-1-4503-4887-4. DOI: 10.1145/3097983.3097987. URL: https://dl.acm.org/doi/10.1145/3097983.3097987 (visited on 04/08/2023).

[54] Grigorios Tsoumakas and Ioannis Katakis. "Multi-Label Classification: An Overview." en. In: *International Journal of Data Warehousing and Mining* (Sept. 2009). DOI: 10.4018/jdwm.2007070101.

[55] Lewis Tunstall, Leandro von Werra, Thomas Wolf, and Aurélien Géron. *Natural language processing with transformers: building language applications with hugging face*. eng. Revised edition. Sebastopol: O'Reilly, 2022. ISBN: 978-1-09-813679-6.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. en. arXiv:1706.03762 [cs]. Dec. 2017. URL: http://arxiv.org/abs/1706.03762 (visited on 03/06/2023).

[57] Yasmen Wahba, Nazim Madhavji, and John Steinbacher. *A Comparison of SVM against Pre-trained Language Models (PLMs) for Text Classification Tasks*. arXiv:2211.02563 [cs]. Nov. 2022. URL: http://arxiv.org/abs/2211.02563 (visited on 02/22/2023).

[58] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. *SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems*. arXiv:1905.00537 [cs]. Feb. 2020. DOI: 10.48550/arXiv.1905.00537. URL: http://arxiv.org/abs/1905.00537 (visited on 04/23/2023).

[59] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. arXiv:1804.07461 [cs]. Feb. 2019. DOI: 10.48550/arXiv.1804.07461. URL: http://arxiv.org/abs/1804.07461 (visited on 04/23/2023).

[60] Zihan Wang, Peiyi Wang, Lianzhe Huang, Xin Sun, and Houfeng Wang. "Incorporating Hierarchy into Text Encoder: a Contrastive Learning Approach for Hierarchical Text Classification." In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 7109–7119. DOI: 10.18653/v1/2022.acl-long.491. URL: https://aclanthology.org/2022.acl-long.491 (visited on 03/23/2023).

[61] Jonatas Wehrmann, Ricardo Cerri, and Rodrigo Barros. "Hierarchical Multi-Label Classification Networks." en. In: *Proceedings of the 35th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2018, pp. 5075–5084. URL: https://proceedings.mlr.press/v80/wehrmann18a.html (visited on 04/04/2023).

[62] Linli Xu, Sijie Teng, Ruoyu Zhao, Junliang Guo, Chi Xiao, Deqiang Jiang, and Bo Ren. "Hierarchical Multi-label Text Classification with Horizontal and Vertical Category Correlations." In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 2459–2468. DOI: 10.18653/v1/2021.emnlp-main.190. URL: https://aclanthology.org/2021.emnlp-main.190 (visited on 03/23/2023).

[63] Hui Ye, Zhiyu Chen, Da-Han Wang, and Brian D. Davison. *Pretrained Generalized Autoregressive Model with Adaptive Probabilistic Label Clusters for Extreme Multi-label Text Classification*. arXiv:2007.02439 [cs, stat]. Aug. 2020. URL: http://arxiv.org/abs/2007.02439 (visited on 02/21/2023).

[64] Abdelhamid Zaidi and Asamh Saleh M. Al Luhayb. "Two Statistical Approaches to Justify the Use of the Logistic Function in Binary Logistic Regression." en. In: *Mathematical Problems in Engineering* 2023 (Apr. 2023). Ed. by Huaiyu Wang, pp. 1–11. ISSN: 1563-5147, 1024-123X. DOI: 10.1155/2023/5525675. URL: https://www.hindawi.com/journals/mpe/2023/5525675/ (visited on 05/24/2023).

[65] Jie Zhou, Chunping Ma, Dingkun Long, Guangwei Xu, Ning Ding, Haoyu Zhang, Pengjun Xie, and Gongshen Liu. "Hierarchy-Aware Global Model for Hierarchical Text Classification." In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 1106–1117. DOI: 10.18653/v1/2020.acl-main.104. URL: https://aclanthology.org/2020.acl-main.104 (visited on 04/06/2023).

Appendix A

Data Understanding - Example of a Research Proposal

- Title: Moral Vagueness in a Mind-Independent World
- URL: https://www.ymner.com/sv/funded-projects/moral-vagueness-in-a-mind-independent-world
- Description: Syfte och mål: The main objective of the project is to show that its central hypothesis holds, namely: that if a leading view in meta-ethics (viz., robust moral realism) is true, then the two leadings theories of vagueness (epistemicism and indeterminacism) have distinct and significant implications for decision-making when those theories are extended to the moral domain. To confirm the main hypothesis, the project proposed several sub-hypotheses. The six months of funding provided by Vinnova allowed to test one of these sub-hypotheses (see below for more on it). Förväntade effekter och resultat: The sub-hypothesis that the six months of funding provided by Vinnova allowed to test was the following: that indeterminacism, despite coming in two versions (a semantic version and a metaphysical version), yields the same account of decision-making under moral vagueness. This sub-hypothesis is the subject matter of the first paper coming out of the project, which is currently at an advance stage of development and being prepared for presentation at Uppsala's Higher Seminar in Theoretical Philosophy. After being presented and revised, it will be submitted to an academic journal. Upplägg och genomförande: The project is split into four Work Packages (the first three for research and the fourth for outreach). One of the research WPs (viz., WP2) is expected

to last twelve months, and it was with this WP that research on the project began. The six months of Vinnova funding were thus spent implementing the first six months of this WP. (The remaining WPs of the project will be implemented, and its remaining sub-hypotheses tested, with the funding provided by the EU, under whose MSCA funding scheme the project will carry on from 2022-05-01 until 2024-04-30.)

Appendix B

FLOPs Calculation

```
# Code based on the paper "ELECTRA: Pre-training Text
↪  Encoders as Discriminators Rather Than Generators"
DROPOUT_FLOPS=4, LAYER_NORM_FLOPS=5, ACTIVATION_FLOPS=8,
↪  SOFTMAX_FLOPS = 5, SIGMOID_FLOPS=4

class TransformerHparams(object):
    """Computes inference FLOPs for transformers."""
    def __init__(self, h, l, s=512, v=30522, e=None, i=None,
    ↪  heads=None, head_size=None, output_frac=0.15625,
    ↪  sparse_embed_lookup=False, decoder=False,
    ↪  num_labels=305):
        self.h = h  # hidden size
        self.l = l  # number of layers
        self.s = s  # sequence length
        self.v = v  # vocab size
        self.e = h if e is None else e  #embedding size
        self.i = h*4 if i is None else i  #intermediate size
        # attn proj sizes
        self.kqv = h if head_size is None else head_size *
        ↪  heads
        # attention heads
        self.heads = max(h // 64, 1) if heads is None else
        ↪  heads
        # percent of tokens using an output softmax
        self.output_frac = output_frac
        # sparse embedding lookups
        self.sparse_embed_lookup = sparse_embed_lookup
        # decoder has extra attn to encoder states
        self.decoder = decoder
        # number of labels
        self.num_labels = num_labels
```

```
def get_block_flops(self):
    """Get the forward-pass FLOPs for a single transformer
    ↪  block."""
    attn_mul = 2 if self.decoder else 1
    block_flops = dict(
        kqv=3 * 2 * self.h * self.kqv * attn_mul,
        kqv_bias=3 * self.kqv * attn_mul,
        attention_scores=2 * self.kqv * self.s * attn_mul,
        attn_softmax=SOFTMAX_FLOPS * self.s * self.heads *
        ↪  attn_mul,
        attention_dropout=DROPOUT_FLOPS * self.s * self.heads
        ↪  * attn_mul,
        attention_scale=self.s * self.heads * attn_mul,
        attention_weighted_avg_values=2 * self.h * self.s *
        ↪  attn_mul,
        attn_output=2 * self.h * self.h * attn_mul,
        attn_output_bias=self.h * attn_mul,
        attn_output_dropout=DROPOUT_FLOPS * self.h *
        ↪  attn_mul,
        attn_output_residual=self.h * attn_mul,
        attn_output_layer_norm=LAYER_NORM_FLOPS * attn_mul,
        intermediate=2 * self.h * self.i,
        intermediate_act=ACTIVATION_FLOPS * self.i,
        intermediate_bias=self.i,
        output=2 * self.h * self.i,
        output_bias=self.h,
        output_dropout=DROPOUT_FLOPS * self.h,
        output_residual=self.h,
        output_layer_norm=LAYER_NORM_FLOPS * self.h,
    )
    return sum(block_flops.values()) * self.s

def get_embedding_flops(self, output=False):
    """Get the forward-pass FLOPs the transformer inputs or
    ↪  output softmax."""
    embedding_flops = {}
    if output or (not self.sparse_embed_lookup):
        embedding_flops["main_multiply"] = 2*self.e*self.v
    # input embedding post-processing
    if not output:
        embedding_flops.update(dict(
            tok_type_and_position=2 * self.e * (self.s + 2),
            add_tok_type_and_position=2 * self.e,
            emb_layer_norm=LAYER_NORM_FLOPS * self.e,
            emb_dropout=DROPOUT_FLOPS * self.e
        ))
    # projection layer if e != h
```

```
    if self.e != self.h or output:
        embedding_flops.update(dict(
            hidden_kernel=2 * self.h * self.e,
            hidden_bias=self.e if output else self.h
        ))
    # extra hidden layer and output softmax
    if output:
        embedding_flops.update(dict(
            hidden_activation=ACTIVATION_FLOPS * self.e,
            hidden_layernorm=LAYER_NORM_FLOPS * self.e,
            output_softmax=SOFTMAX_FLOPS * self.v,
            output_target_word=2 * self.v
        ))
        return self.output_frac *
        ↪  sum(embedding_flops.values()) * self.s
    return sum(embedding_flops.values()) * self.s

def get_binary_classification_flops(self):
    classification_flops = dict(
        hidden=2 * self.h * self.h,
        hidden_bias=self.h,
        hidden_act=ACTIVATION_FLOPS * self.h,
        logits=2 * self.h
    )
    return sum(classification_flops.values()) * self.s

def get_multi_label_classification_flops(self):
    classification_flops = dict(
        hidden=2 * self.h * self.h,
        hidden_bias=self.h,
        hidden_act=ACTIVATION_FLOPS * self.h,
        logits= self.num_labels * self.h,
        sigmoid_flops = SIGMOID_FLOPS * self.num_labels
    )
    return sum(classification_flops.values()) * self.s

def get_infer_flops_multi_label(self):
    """Get the FLOPs for running inference with the
    ↪  transformer on a multi-label
    classification task."""
    if include_embedding:
            embedding_flops =
            ↪  self.get_embedding_flops(output=False)
        else:
            embedding_flops = 0

        return ((self.l * self.get_block_flops()) +
```

```
                embedding_flops +
                self.get_multi_label_classification_flops())
```

Appendix C

F1-Score Averages

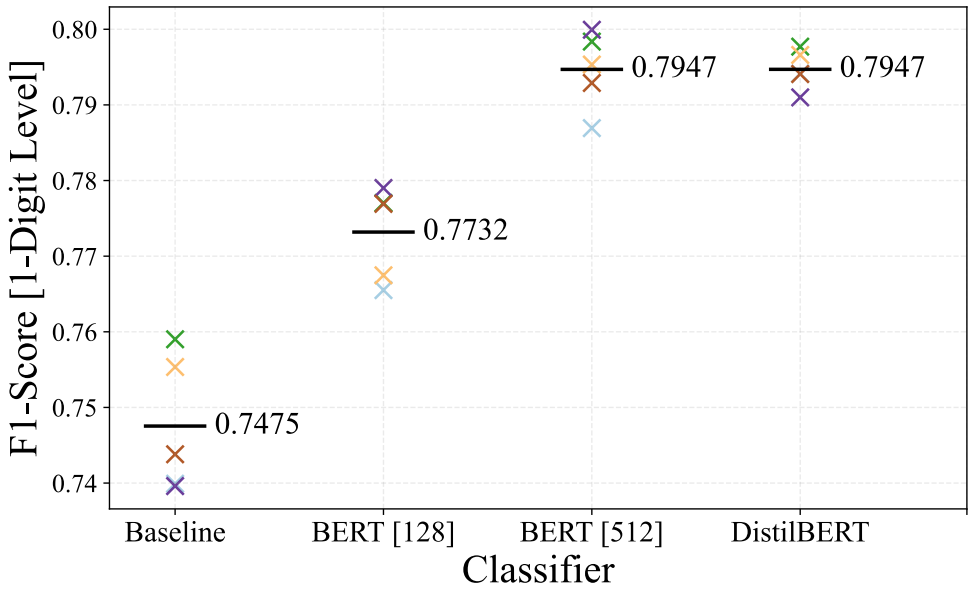


Figure C.1: F1-score for the labels of the broadest classification level that is represented by a label ID consisting of one digit. Each × displays one out of five random seeds. The mean is illustrated as the horizontal black line.

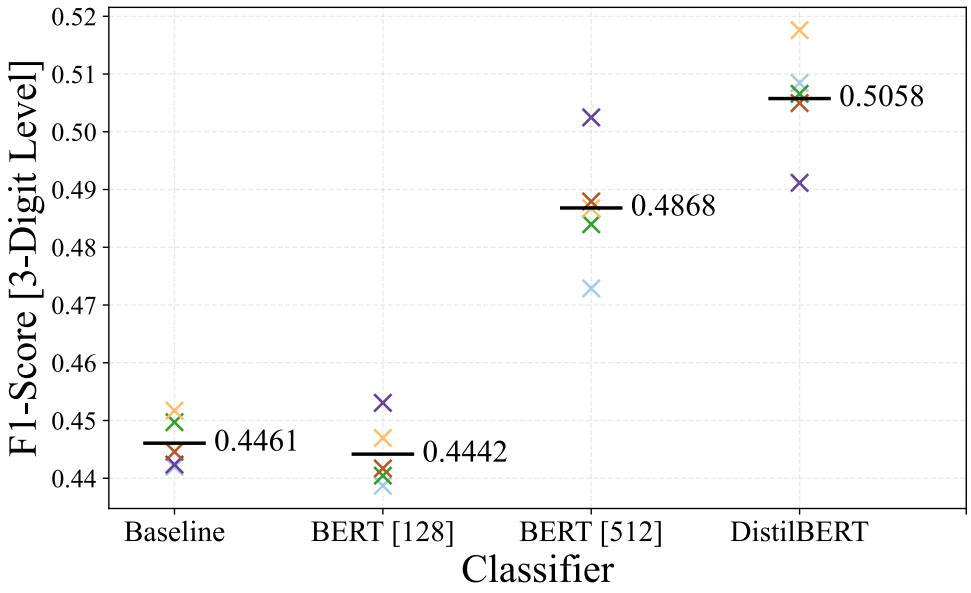


Figure C.2: F1-score for the labels of the classification level that is represented by a label ID consisting of three digits.

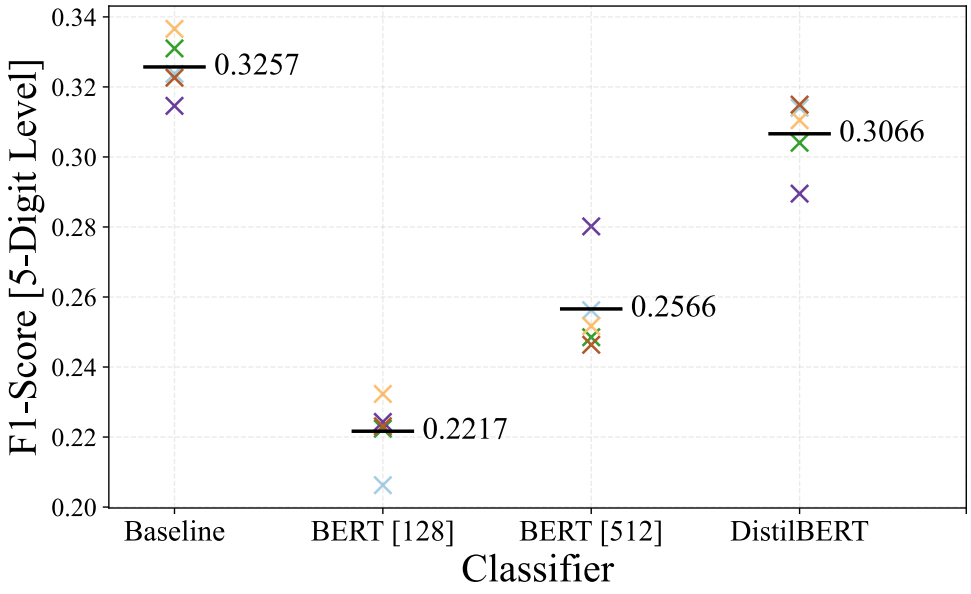


Figure C.3: F1-score for the labels of the most specific classification level that is represented by a label ID consisting of five digits.

Appendix D

Comparison Performance and Usage Cost

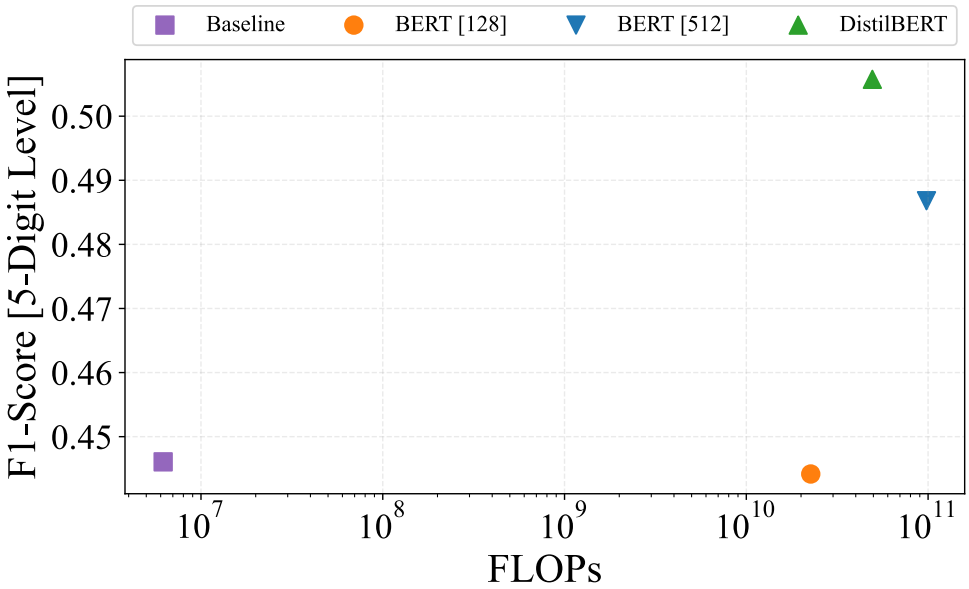


Figure D.1: Comparison of usage cost and performance averaged on the test sets regarding the 3-digit classification level for each classifier. Here, the FLOPs are used without including the embedding and are scaled logarithmically.

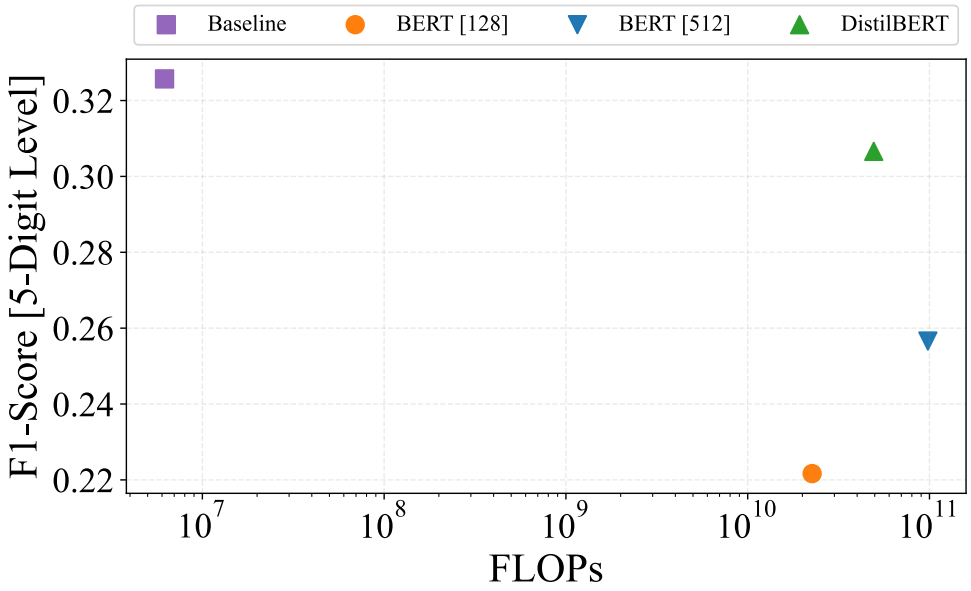


Figure D.2: Comparison of usage cost and performance averaged on the test sets regarding the 5-digit classification level for each classifier. Here, the FLOPs are used without including the embedding and are scaled logarithmically.

TRITA– XXX-XXX 20XX:XX